NASA/CR-1998-206915

Formal Specification of a Flight Guidance System

Francis Fung and Damir Jamsek
Odyssey Research Associates, Ithaca, New York

January 1998

NASA/CR-1998-206915

Formal Specification of a Flight Guidance System

Francis Fung and Damir Jamsek
Odyssey Research Associates, Ithaca, New York

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-20335

January 1998

Contents

Chapter 1

Introduction

This document contains a formal specification for the mode logic of a Flight Guidance System (FGS) and a discussion of issues raised by writing the formal specification. A flight guidance system is an example of life-critical code; this project aims to demonstrate the effectiveness of formal methods in the requirements analysis and design of life-critical systems. The source document for our project is “Specifying the Mode Logic of a Flight Guidance System in CoRE,” by Stephen Miller and Karl Hoech, Rockwell-Collins.

1.1 Introduction to the Flight Guidance System

Most modern aircraft possess a computerized system, called a Flight Guidance System (FGS), that uses commands from the pilots to select flight control laws for the aircraft. The resulting flight commands can then be used as advisories for the pilots, or the advisories can be routed to an autopilot which will automatically guide the aircraft.

An FGS consists of two broad parts: the mode logic and the flight control laws. The mode logic comprises the possible settings of the FGS; it is altered by events such as pilot commands or changes in monitored variables. The FGS reads the mode of the mode logic and then invokes the corresponding set of flight control laws to generate pitch and roll commands to guide the aircraft. These commands can be taken as advisories to the pilot; or, if the autopilot is turned on, the commands can be executed by the aircraft without any human intervention.

The work on this task is based on the source document “Specifying the Mode Logic of a Flight Guidance System in CoRE,” by Stephen Miller and Karl Hoech, which gives a description of a simplified specification of a Flight Guidance System (FGS), using the CoRE specification method. The CoRE FGS specification is designed to exhibit many of the difficulties that arise in specifying an event-driven system such as an FGS. In particular, their model of the FGS is decomposed into several concurrent (and hierarchical) state machines that are allowed to influence each other.

Miller and Hoech’s specification deals only with normal behavior of the system, and does not discuss how to deal with internal errors, such as component failures. The authors have released this specification as a resource, on which to test various methods and tools. They state that their specification undoubtedly has errors; this, however, enhances its value as a example on which to compare the results of various methodologies. In addition, they propose and use certain extensions to the CoRE method, some of which they define only informally. These extensions provide another source of questions and comparisons.

In this report, we construct a formal Z specification based on the CoRE FGS. There are at least two inequivalent ways of formalizing the semantics of the source document, depending on whether we use micro-time semantics or not. In order to translate the document into Z, we have chosen to interpret the CoRE FGS using, to the extent possible, the SCR discrete-time formal model [3]. In particular, we have not used any micro-time semantics. This approach is compatible with the Z philosophy of specifying states and transitions. A complementary investigation of the CoRE FGS using micro-time semantics appears in Naydich and Nowakowski [6].

The CoRE notation is very readable, and we are not claiming that the Z notation is more suited for specifying this FGS. We are using the Z notation here to express a semantic definition for the FGS, and also as input to the Z/EVES theorem prover [7, 8, 9].

Because we were only given the paper specification and we did not have formal definitions of the several aspects of the CoRE FGS, we have translated the specification by hand. However, the translation could certainly be automated.

1.2 The Use of Formal Methods in Software Specification

A *formal specification* of a system is a description given in a language that possesses a formally defined and unambiguous syntax and semantics. The Z language [4, 10, 11] is an example of such a language. A formal specification is thus a precisely described constraint on the system's behavior. The process of formulating a formal specification of a system is useful in detecting ambiguities and design flaws at an early stage in the life cycle of a software product.

A formal specification allows us to perform certain kinds of type checks and consistency checks on the specification. We may also desire to prove that certain invariants hold for all states of the system. We can use automated tool support, such as Z/EVES [7, 8], to aid us in proving consistency checks and invariants.

Once we have constructed an algorithm that implements the system, it is desirable to generate a *formal verification* that the algorithm indeed satisfies the requirements of the specification. Formal verification applies to all inputs to the system, and thus complements conventional trial-and-error debugging methods, which can usually test only a certain sampling of the inputs.

1.3 Structure of This Document

Our report is structured as follows. Chapter 2 is an outline of the CoRE method, which was used by Miller and Hoech to construct the source document. The canonical reference on CoRE is the *Consortium Requirements Engineering Guidebook* [1] (or "CoRE Guidebook") by Faulk, Finneran, Kirby and Moini. Chapter 3 is an introduction to the Flight Guidance System (FGS) that is specified in Miller and Hoech's report [5]. Chapter 4 is a discussion of some questions that arose from our investigations of the CoRE specification. Chapter 5 contains an outline of the Z specification language; for further reading, there are two books by Spivey: *Understanding Z* [10] and *The Z Reference Manual* [11], as well as Jacky's *The Way of Z* [4]. Chapter 6 discusses our work in constructing a formal specification of the FGS in Z. Chapter 7 contains conclusions and acknowledgments. Appendix A contains the formal specification of the FGS in Z.

Chapter 2

Preliminaries on CoRE

2.1 Introduction

The CoRE (Consortium for Requirements Engineering) method [1], which is promoted by the Software Productivity Consortium (SPC), is a method for developing and writing software requirements in a reasonably formal yet readable manner. CoRE is a close relative of the Software Cost Reduction (SCR) method [3]. The CoRE Guidebook [1] provides a detailed explanation of how to construct a CoRE specification.

We will give a brief summary of the CoRE method (see [1] for more details). First, CoRE is built on Parnas' *four-variable model* of embedded-system behavior. In this model, a system is viewed as interacting with the external world. The system monitors certain external quantities (called *monitored variables*) and controls the values of other external quantities (called *controlled variables*). The system is specified as a collection of relations between the monitored variables and the controlled variables.

The monitored variables are external quantities, such as air pressure and altitude. From these, the system derives inputs to the software itself; for instance, a sensor on the outside of the plane measures a monitored quantity and then sends a binary number to the computer. The inputs to the software itself, such as this binary number, are called *input variables*.

The system, based on the values of the input variables, generates values for the *output variables*. These may again be binary numbers, which are then converted into values of the controlled variables; for instance, the display of the air pressure in the cockpit is a controlled variable.

The crux of the four-variable model is that we should specify the behavior of the system in terms of relations between the monitored and the controlled variables, rather than in terms of inputs and outputs to the software itself. The four-variable model encourages us to write a high-level specification of the system behavior, rather than attempting to implement design or hardware-level decisions.

Once we have isolated the monitored and controlled variables, we must determine the constraints on their values. Some of the constraints on the variables are imposed by environmental constraints. For instance, the possible speed of an aircraft is limited by the mechanical properties of the aircraft. Such constraints constitute the *NAT relations* on the variables, since they are "natural" or external constraints on the values of the variables.

Once we have found the NAT relations, we must isolate the relations between the monitored and controlled variables that are to be imposed by the system itself. For instance, a gauge on a flight control panel may be required by the system to display the altitude of the airplane, rounded off to the nearest hundred feet. These relations, which the software "requires" of the variables, are called the *REQ relations*.

Finally, we must determine the relationship of the monitored variables to the inputs variables (the *IN relations*) and the relationship of the output variables to the controlled variables (the *OUT relations*). These are, of course, design-level decisions, based on the nature of the components used to construct the system. We can analyze the REQ and NAT relations independently of the IN and OUT relations. The specification of the REQ and NAT relations of the system constitutes the *CoRE behavioral model* of the system.

CoRE builds an additional model, the *CoRE class model*, on top of the behavioral model, by arranging the variables into classes in an object-oriented fashion. For each class, certain variables are visible to other classes and can thus be *exported*.

Often, it is the case that the same expression involving monitored variables appears several times in the specification. As a shorthand, we can introduce *term* variables which abbreviate expressions of monitored variables, system modes, or other terms.

A *state* of the system consists of an assignment of a value to each variable of the system. The state of the system changes exactly when the value of some variable changes. A change of a variable's value constitutes an *event*.

The system's behavior may depend not only on the present state of the system, but also on the previous values, or *state history*, of the system. The system can capture relevant state history information using certain kinds of finite state machines called *mode machines*. In CoRE, a *mode machine* consists of a finite set of states called *modes*, a distinguished initial mode, a set of transition events, and a set of mode transitions that express the effect of the events on the mode machine.

Thus, a mode machine is a finite state machine such that the behavior of the machine is defined entirely in terms of the CoRE behavioral model, and also such that the machine does not actually perform any actions, but only records state information. It is conceivable that a finite state machine can have a command such as "drill the rock" as a state; but the mode machines in CoRE are not allowed to behave in this fashion. The modes are merely information, which can be read by other aspects of the system.

2.2 Tables

Much of a CoRE specification consists of *tables*. A table specifies the value of a variable according to values of other variables and possibly events. There are three kinds of tables: condition, event, and mode transition.

2.2.1 Condition Tables

A *condition table* for a variable gives a partition of the system state into mutually exclusive conditions, and a value of the variable for each condition. The rows are usually used to partition the possible states according to the mode values of one or more system modes. Note that every system state must satisfy one, and only one, of the conditions in the table. For instance,

Condition Table for `CondVar`			
`Mode = A`	`Var1 = T`	`Var1 = F`	`X`
`Mode = B`	`X`	`X`	`TRUE`
`CondVar`	`Val1`	`Val2`	`Val3`

gives the value of the variable `CondVar` in terms of the values of the values of `Mode` and `Var1`. If `Mode = A`, then if `Var1 = T`, then `CondVar = Val1`; otherwise, if `Var1 = F` then `CondVar = Val2`. The `X` means that if `Mode = A` then no condition can make `CondVar` equal to `Val3`. The second row, the `TRUE` means that when `Mode = B`, there is no additional condition that must be satisfied to make `CondVar` equal to `Val3`.

2.2.2 Event Tables

An *event table* for a variable tells us which events cause the variable to change, and tells us the possible new values of the variable. As with condition tables, the rows are usually used to partition the possible states according to the values of one or more mode machines.

Event Table for `EventVar`			
`Mode = A`	`Event1`	`Event2`	`X`
`Mode = B`	`Event2`	`Event1`	`Event3`
`EventVar`	`Val1`	`Val2`	`Val3`

The first row of this table means that if `Mode = A`, then the occurrence of `Event1` will cause `EventVar` changes to `Val1`, and the occurrence of `Event2` will cause `EventVar` changes to `Val2`. The second row has a similar meaning. It is implicit that, if no listed event occurs, the variable's value does not change.

2.2.3 Transition Tables

A *transition table* for a mode machine gives the possible transitions of that mode machine. For each mode of the machine, the table gives events that can change the mode, and also the resulting mode of the machine after the event has occurred.

Transition Table for `Mode`		
From	Event	To
`A`	`Event1`	`B`
`A`	`Event2`	`C`
`B`	`Event2`	`C`
`C`	`Event1`	`A`

The first line says that if `Mode = A` and `Event1` occurs, then `Mode` changes to `B`. The other rows have similar interpretations. It is also implicit in this table that if no listed event occurs, then the value of `Mode` does not change.

2.2.4 Table Properties

An essential property of a table is that it be *disjoint*. For condition tables, this means that in each row, the conditions listed must be mutually exclusive. For event or mode transition tables, this means that for each mode value, no two of the listed events can occur simultaneously. As well, condition tables must be *complete*; this means that all possible states of the system satisfy one of the listed conditions.

2.3 Remarks on CoRE Semantics

The variables in CoRE are considered as functions of (continuous) time. Events in CoRE are assumed to occur instantly, and to be atomic (indivisible). Note that the change of any variable (not just a monitored variable) is an event and can be used to initiate another transition. The continuous nature of the CoRE variables can lead to some questions about the timing of events that are created by other events, since the CoRE model supposes that they happen simultaneously.

The CoRE Guidebook leaves several issues, such as timing of events, up to the authors of a specification. One resolution is to use CoRE with the formal semantic model of SCR [3]. This model uses discrete time polling cycles, rather than continuous time, and gives interpretations for tables and so forth. A key feature of the SCR formal model is a partial ordering on the variables to simulate one variable being dependent on another. The existence of such a partial order guarantees that we can linearly order the updating of the variables in such a way that the new value of a given variable depends only on old values of variables and new values of previously updated variables. Given the SCR formal model, there is a set of static consistency checks (detailed in [3]) that can be performed on the tables of the specification. These checks guarantee complete and deterministic behavior of the system (that is, for each old system state and input event that can occur, there is one, and only one, new system state that can result).

The authors of the CoRE FGS specification found the restrictions of the SCR formal model inconvenient when they wrote their specification. In particular, they wanted to use concurrent mode machines that could trigger transitions in each other; such machines cannot be partially ordered by dependency.

In addition, the authors introduce several concepts that are not formally defined in CoRE (some of which are explicitly prohibited in the SCR formal semantic model). In particular, the CoRE guidebook does not address how to deal with simultaneous events (which, when interpreted in certain ways, can lead to "cascading" internal transitions), "continuously occurring" events, or concurrent mode machines that are allowed to drive each other, all of which play large roles in the CoRE FGS specification. The authors of the CoRE FGS specification have written some informal semantics for these concepts. Therefore, in order to provide a formal specification for the FGS, a formal semantics must be chosen for these concepts.

Chapter 3

The Flight Guidance System

3.1 Outline of the FGS

Here is an outline of the Flight Guidance System. An aircraft can move about three axes: lateral (roll), vertical (pitch) and side-to-side (yaw). The mode logic of the FGS is divided into a number of mode machines; there is one for the lateral direction, and another for the vertical direction. (The yaw control has been omitted from the CoRE FGS example for simplicity.) Two other mode machines control the acquisition of a preselected altitude, and the vertical approach to a landing. For each state, the FGS invokes *flight control laws* that generate commands for the aircraft itself. For instance, if the lateral mode is in a certain mode called `HDG`, then the FGS uses the corresponding flight control laws to generate commands that will put the plane on a preselected heading.

These commands are either taken as advisories to the pilots (which are annunciated by the *Flight Director*) or, if the *Autopilot* is activated, they are passed directly to the autopilot to control the plane's motion.

For the most part, the modes are selected by the pilots, using *switches* on the *flight control panel* or a number of other controls. There are also some environmental events that can change a mode, such as when the plane exceeds a certain speed.

3.2 The Mode Machines

The first four mode machines that we mention are the `mode_Active_Lateral`, `mode_Active_-Vertical`, `mode_Altitude_Select` and `mode_Vertical_Approach` mode machines. Together, these four determine the selection of the flight control laws. The other mode machines of the FGS control the flow and annunciation of data.

Each of the mode machines is hierarchical, in the sense that some mode values have submodes. Because CoRE allows one to specify what values to export, some mode values may also be considered submodes of a "supermode" mode value, which can be used in transitions.

The `mode_Active_Lateral` mode machine selects the flight control laws controlling the aircraft in the horizontal plane, through the generation of roll commands. It can be in one of the following modes: `ROLL, HDG, NAV, APPR, or GA`. Each of `NAV` and `APPR` has two submodes, `Armed` and `Track.` As well, `ROLL` has two submodes, `Hdg_Hold` and `Roll_Hold`.

The `mode_Active_Vertical` mode machine selects the flight control laws controlling the aircraft in the vertical plane, through the generation of pitch commands. It can be in one of the following modes: PITCH, FLC, VS, ALTSEL, ALTHOLD, APPR, GA.

The mode_Altitude_Select mode machine describes the logic used to capture and track a preselected altitude. It can be in one of the following modes: CLEARED, ARMED, or ACTIVE. There is an ENABLED "supermode" that comprises ARMED and ACTIVE. The ACTIVE mode has two submodes: Capture and Track.

The mode_Vertical_Approach mode machine describes the logic use to capture and track a precision vertical approach. It can be in one of the following modes: CLEARED, ARMED, and TRACK. There is an ENABLED "supermode" that comprises Armed and Track.

The mode machines are not independent; in fact, certain transitions of one mode machine are initiated by a mode transition of a different mode machine. Only certain combinations of modes are possible; for instance, mode_Altitude_Select must be CLEARED whenever mode_Active_Vertical is in one of the modes APPR, GA, or ALTHOLD.

The mode machine mode_Autopilot controls whether the autopilot is automatically applying the flight control laws to the aircraft. It can be in one of the following modes: ENGAGED, DISENGAGED, or DISENGAGED_WARNING.

The mode machine mode_Flight_Director controls annunciation of FGS-generated flight commands. It can be in one of the following modes: ON or OFF. The ON mode has two submodes: CUES and NO_CUES.

The mode machine mode_Overspeed records whether the aircraft is going too quickly. It can be in one of the following modes: SPEED_OK or TOO_FAST.

The FGS interacts with several systems, such as the Flight Control Panel, Control Yokes, Throttles, Air Data Computer, navigation sources, and several displays. The Flight Control Panel, Control Yokes, and Throttles comprise knobs and switches to select system modes and to set references such as desired heading and airspeed. The Air Data Computer provides data about the measured state of the aircraft, such as airspeed and altitude.

Chapter 4

Questions Arising from the FGS Specification

4.1 Introduction

The CoRE specification of the Flight Guidance System has several mode machines, which are allowed to influence each other's behavior. In particular, a change in the value of a variable such as the mode `mode_Active_Vertical` is considered an event, and can be used to trigger other transitions.

As we described above, we chose a formal semantic model for the CoRE specification based on the SCR discrete-time formal model. Using this model, we find several difficulties in the CoRE specification as written. For each one, we have to discuss whether the error arises because our model does not quite capture the intent of the authors of the CoRE specification, or because of an actual error in the CoRE specification.

The major issues that we found are discussed below. We have given ideas for possible resolutions where feasible. Our goal in this project is to translate these concepts using SCR discrete-time semantics, of which the main ideas are as follows. An implementation of an event-driven system is very likely to be implemented as a discrete-time entity, where the state of the system is known only at regularly spaced intervals. At each interval, we *poll* the values of each variable; thus the interval can be referred to as a *polling cycle*. The state of the system at the beginning of the cycle is the *old* state, and the state at the end is the *new* state. We will use `typewriter` variables for variables in the CoRE specification, and *italic* variables for variables in our discrete-time translation. By convention, for any variable Var of the old state, we denote by Var' the corresponding variable for the new state.

Once we have chosen these interpretations of event and transition, then "two events are simultaneous" means that the two variables change in the same polling cycle.

4.2 Simultaneous Events

A Flight Guidance System is an example of an event-driven system; that is, the system reacts to events by changing its state. A fundamental question for any such system is how to deal with simultaneous events.

First, an *input event* is a change in a monitored variable. These are the only events that are allowed to instigate a change in the state of the system. There will always be physical constraints in an actual implementation as to how quickly a system can process input events. So, there will always be some question of what to do when the system cannot tell whether one input event is prior to another. Thus either we must make the assumption that all external events can be separated in time, or we must specify how the system behaves when confronted with simultaneous events.

One option is to assume that the system acts nondeterministically and randomly picks the order in which to process the input events. This is, of course, not desirable if we want predictability in our system.

Another option is to attempt to formulate responses for all combinations of input events. This rapidly becomes messy and impractical, especially because the number of responses that we have to specify will grow exponentially with the number of input events happening simultaneously. Another option is to formulate a priority hierarchy for the input events. This may be feasible, but it can be argued that we can separate the specification of this aspect of the system from the rest of the specification.

Investigators of SCR, a close CoRE relative, have studied the so-called One Input Assumption, where one assumes that only one input event can happen at a given time. We have chosen to use this assumption. We can assume, if necessary, that we will give a separate specification of a "preprocessor" which would take simultaneous events and, by some method such as one of the options above, construct a sequence of non-simultaneous events to feed to the system. Even with the One Input Assumption, we still face questions concerning simultaneous internal events, because changes of internal variables are also events and can be used to drive mode machines.

Each event in CoRE can be represented as a change in a Boolean variable `Bool` from `FALSE` to `TRUE`, or from `TRUE` to `FALSE`. The corresponding events are called `@T(Bool)` and `@F(Bool)`. The Boolean `Bool` could very well be a Boolean expressing whether one of the mode machines is in a certain mode.

Suppose the event `@T(Bool1)` causes `Bool2` to change from `FALSE` to `TRUE`, therefore spawning `@T(Bool2)`. If these are to be considered events, then does `@T(Bool1)` occur "before" `@T(Bool2)`? Perhaps there is a machine whose transitions are triggered by both `@T(Bool1)` and `@T(Bool2)`. To which event does this machine respond?

For example, consider the following three transitions taken from the CoRE specification of the FGS. The notation $\overline{\texttt{HDG}}$ means "any mode value except `HDG`".

From the `mode_Active_Lateral` transition table:

Number	From	Event	To
27	`GA`	`@F(mode_Active_Vertical = GA)`	`ROLL`
28	$\overline{\texttt{HDG}}$	`@HDG_Switch_Pressed`	`HDG`

From the `mode_Active_Vertical` transition table:

Number	From	Event	To
57	`GA`	`@F(mode_Active_Lateral = GA)`	`PITCH`

Suppose that `mode_Active_Lateral = GA` and `mode_Active_Vertical = GA`. Then suppose the `HDG` switch is pushed; by Transition 28, `mode_Active_Lateral` mode switches from `GA` to `HDG`. This creates the event `@F(mode_Active_Lateral = GA)`, which by Transition 57 causes `mode_-Active_Vertical` to change from `GA` to `PITCH`. But this creates the event `@F(mode_Active_Vertical = GA)`. If these three events (`@HDG_Switch_Pressed`, `@F(mode_Active_Lateral = GA)`, `@F(mode_-Active_Vertical = GA)` are assumed to occur simultaneously, then then `mode_Active_Lateral` is asked simultaneously to change from `GA` to `ROLL`, and from `GA` to `HDG`.

In the semantics we have chosen for the Z specification, the behavior in the previous paragraph is inconsistent. There are other methods (such as RSML) whose semantics support "micro-time" between the external events, in which the internal events occur in sequence, so the above "cascade" would be considered valid. The CoRE guidebook is silent on the issue of micro-time.

The authors of the FGS wrote that they had chosen an interpretation where internal events happen in sequence. They state (p. 20) that "if more than one chain is possible, a chain is selected non-deterministically"; apparently this means that if multiple conflicting transitions are initiated then one is chosen at random. However, they do not formulate a precise semantics for their interpretation; in particular, there are no rules for ordering the spawned internal events into a sequence. One can easily provide examples where using different rules would give rise to different system behavior. For instance, we could resolve events breadth-first, where the input event spawns several events, and then each of these is resolved before any of the events that they spawn are considered; or we could resolve in some depth-first manner, where each event and all its consequences are resolved before the next event is considered.

Without formal definitions or semantics for internal events, it is very difficult to resolve any ambiguities because different users of the specification can easily hold contradictory interpretations, and may have difficulty Communicating with each other unless these interpretations are made explicit. A formal specification is therefore very useful in establishing a common and unambiguous frame of reference for all users of the specification.

4.3 Possible Resolutions

Since we do not assume any notion of micro-time, the three events given above would happen simultaneously, and thus Transitions 27 and 28 would be nondisjoint. A method to resolve this difficulty is to restrict Transitions 28 and 57 so that they can never occur simultaneously.

It appears that the intent of Transition 28 is that `mode_Active_Vertical` and `mode_Active_Lateral` should always leave the mode `GA` together, and if an event occurs that "directly" forces `mode_Active_Vertical` to leave `GA` but the event does not "directly" affect `mode_Active_Lateral`, then `mode_Active_Lateral` should go to `ROLL`. Of course, the analogous statement holds for events that "affect only `mode_Active_Lateral` directly".

We can make the notion of "an event directly affecting a mode" explicit as follows. We begin by collecting a list of input events. Because of the One Input Assumption, only one input event can occur in any given polling cycle. Given a list of the possible input events (which we construct for our Z formal specification), then we define a subset `Vertical_Events` of the input events that we consider to directly affect the `mode_Active_Vertical` mode machine; it will certainly include those input events that appear in the `mode_Active_Vertical` mode transition table, as well as some of the input events that cause internal events that appear in the `mode_Active_Vertical` mode transition table.

Then we can translate the event that triggers Transition 57 to be "`@F(mode_Active_Lateral= GA)` and the input event is in the set `Lateral_Events` but not in the set `Vertical_Events`".

For instance, we would put `HDG_Switch_Pressed` in the `Lateral_Events` subset, but not in the `Vertical_Events` subset. Then the modified Transition 28, whose event trigger is "`@F(mode_Active_Vertical) = GA` and the input event is in the set `Vertical_Events` but not in the set `Lateral_Events`" would not be triggered.

Modifying the transitions in this manner effectively forces them to be mutually exclusive, since the two sets of input events associated to the modified transitions are disjoint.

Since the two subsets are quite close to the same information carried in the transition tables in the CoRE specification, this construction is not so artificial. However, if the transition tables are heavily dependent on internal events, it may require some effort to decide which external events directly affect a mode machine, and which should be excluded.

4.4 Continuous Transitions

Another controversial aspect of the FGS CoRE specification is the use of so-called "continuous" actions and mode transitions. The authors state that this has a simple and intuitive meaning. However, they do not give a formal definition of the notion in the CoRE specification.

For instance, the event table defining the variable term_Reference_IAS is as follows, where (**FD** = mode_Flight_Director, **AV** =mode_Active_Vertical).

FD	**AV**	Event	
OFF	N/A	X	X
ON	$\overline{\mathtt{FLC}}$	X	@Speed_Knob_Changed
ON	FLC	ENTERED	@Speed_Knob_Changed
		CONTINUOUSLY WHEN term_SYNC	
term_Reference_IAS		mon_Indicated_Airspeed	limit(0, 512, term_Reference_IAS′ + 1*(term_Speed_Knob_Rotation))

First, the variable term_Reference_IAS can change only when mode_Flight_Director = ON. Then term_Reference_IAS is required to be "continuously" equal to the monitored mon_Indicated_Airspeed whenever mode_Active_Vertical= FLC and term_SYNC = TRUE). However, the other part of the definition of term_Reference_IAS says that the variable term_Reference_IAS also changes, according to a certain formula, when the Speed_Knob is changed. Since term_SYNC corresponds to a button being held down, it is possible that the crew will try to change the Speed knob while term_SYNC = TRUE. Without a formal definition of "continuously" we cannot say whether this event table is nondisjoint. (The authors of the CoRE specification had already stated this in their document.)

One would like to capture the continuous event as follows: whenever mode_Flight_Director = ON, mode_Active_Vertical = FLC and term_SYNC = TRUE, we want to set term_Reference_IAS equal to mon_Indicated_Airspeed. This is akin to a condition table. To capture the second condition, we would say that in a polling cycle where the Speed Knob is changed, we give term_Reference_IAS the value given by the formula in the table. But this is a feature of an event table. Since these two types of tables have rather different interpretations in a polling cycle model, we have to find a way to reconcile them.

We have chosen to view term_SYNC = TRUE as overriding the @Speed_Knob_Changed event. However, since we want the effect to be "continuous," we use a different interpretation from that of the "when" guard. The One Input Assumption guarantees that @Speed_Knob_Changed will never occur simultaneously with @T(term_SYNC), so there will be no conflict. We will discuss more details of our translation in a later section on translating event tables into Z.

A formal interpretation of "continuous transition" also seems slippery. (Let us note that the authors of the CoRE specification introduced this concept specifically to generate debate about the various methods of defining it.)

Let us give an example from the mode_Active_Vertical transition table. Transition 52 of the CoRE specification reads

Number	From	Event	To
52	$\overline{\texttt{FLC, APPR, ALTHOLD, ALTSEL}}$	CONTINUOUSLY WHEN term_Overspeed	FLC

The term term_Overspeed is a Boolean which is TRUE exactly when the aircraft is exceeding the maximum safe operating speed. This transition is intended to force the plane to stay in one of the modes FLC, APPR, ALTHOLD, or ALTSEL; approximately, if any actions causes mode_Active_Vertical to enter some mode other than FLC, APPR, ALTHOLD, or ALTSEL, then mode_Active_Vertical should immediately be switched into FLC.

An informal interpretation of this "continuous transition," which the authors of the CoRE specification seem to support, is that if mode_Active_Vertical is switched out of the set of mode { FLC, APPR, ALTHOLD, ALTSEL }, (say for example to GA) then mode_Active_Vertical mode is "as soon as possible" switched to FLC. This interpretation raises several questions.

- The first question is: for how long is mode_Active_Vertical actually in the other mode before switching into FLC?

  Here is a sample scenario. Suppose mode_Active_Vertical is in ALTSEL, mode_Active_-Lateral is in HDG, and term_Overspeed = true. Now suppose the GA switch is pressed.

  If we model the transitions by polling cycles, then does mode_Active_Vertical actually change to the other mode GA for a cycle and then change again to FLC? Or does mode_Active_Vertical switch out of ALTSEL directly into FLC?

- This raises another interesting question: what effect does this continuous transition have on the other modes, such as mode_Active_Lateral? Suppose mode_Active_Vertical enters the mode GA for one polling cycle when the GA switch is pressed. Now, according to Transitions 31 and 55, when the GA switch is pressed, mode_Active_Vertical and mode_Active_Lateral both switch to GA. Then mode_Active_Vertical is switched "as soon as possible" to FLC. When mode_Active_Vertical mode is switched out of GA, then, by Transition 27, mode_-Active_Lateral mode switches to ROLL. (Note that this continuous transition would certainly not be included in our proposed Lateral_Events set, so a modified Transition 27 would still apply here).

- Now suppose that mode_Active_Vertical mode is switched directly to FLC when the GA switch is pushed. Then does the mode_Active_Lateral mode switch directly to ROLL as a result, even though neither mode_Active_Vertical or mode_Active_Lateral was actually in GA at all? Or should mode_Active_Lateral simply stay in HDG mode?

  The authors of the CoRE specification (personal communication) say that they consider mode_Active_Vertical to enter and leave the GA mode very briefly. However, this does not tell us which of the above discrete-time polling cycle models best captures their intention. When asked whether mode_Active_Lateral should stay in HDG, or switch to ROLL, in the above scenario, they replied that a case could be made for either. Since this is a question at the customer requirements level, we cannot answer it here. However, it is clearly a question that an actual specification, in order to be unambiguous, must be able to answer.

4.5 Possible Resolutions

One might be tempted to simply include some sort of `IF-THEN-ELSE` clause at the beginning of the `mode_Active_Vertical` transition table, such as "if `term_SYNC = TRUE` then (some effects) else (normal behavior of `mode_Active_Vertical`)". Or, similarly, for every transition of `mode_Active_Vertical` into a mode which is not one of `APPR, ALTSEL, ALTHOLD`, or `FLC`, we could split the transition into two parts. When `term_Overspeed = FALSE`, then `mode_Active_Vertical` ends up in the desired mode. When `term_Overspeed = TRUE`, then `mode_Active_Vertical` ends up in `FLC`. This is of course a rather inelegant approach, since there could easily be a large number of such transitions, and there ought to be a convenient way of expressing this behavior succinctly. The benefit of this approach is that it fits very strictly into the polling cycle framework.

However, the `mode_Active_Vertical` mode machine cannot be considered in isolation. We would also have to modify transitions in other tables, such as the effect of `@GA_Pressed` on `mode_Active_-Lateral = HDG`. Since one of the intended invariants of the specification is that `mode_Active_Lateral` is in `GA` only when `mode_Active_Vertical` is in `GA`, we must then send `mode_Active_Lateral` to a different mode from `GA` if we send `mode_Active_Vertical` to a different mode ffrom `GA`.

If we assume that the intention is that `mode_Active_Vertical` indeed enters the `GA` mode momentarily, then we are faced with another difficulty. Suppose that we implement the mode transition so that, when `term_Overspeed = TRUE` and `mode_Active_Vertical` is switched out of ({ `APPR, ALTSEL, ALTHOLD, FLC`}), then `mode_Active_Vertical` is then switched back to `FLC`. Then the switch to `FLC` really must be considered as happening "after" the original switch. This leads us back to the lack of "micro-time" semantics.

We believe that a crucial advantage of CoRE, as interpreted using the SCR discrete-time model, is that when a transition is specified (with an old mode value, event, and new mode value), the transition indeed gives the stated relation between the old state and the new state. In order to introduce micro-time semantics, we would have to give up this advantage entirely. We do not see any easy interpretation of a continuous transition in our formal semantic model, without a drastic rearrangement of the transitions as described above.

Because we have interpreted transitions such as Transition 55 above to actually send `mode_-Active_Vertical` to `GA`, we cannot guarantee the CoRE FGS Invariant 11, which requires that (`term_Overspeed = TRUE` $\Rightarrow$ `mode_Active_Vertical` $\in$ {`ALTSEL, ALTHOLD, APPR, FLC_Overspeed`}). Interestingly enough, the authors of the CoRE FGS specification suggest (personal communication) that their intent is that `mode_Active_Vertical` goes to `GA` momentarily during the continuous transition. Thus their specification cannot guarantee this invariant either, unless they define a semantics in which their invariant only holds "once all of the internal events have resolved" (which of course would also require a formal definition).

Using micro-time opens up a new range of questions. Once you admit multiple steps in a single transition, then you have to choose whether to allow only a fixed number, or you must must be prepared to check for infinite cascades and loops. You must also define an order in which to evaluate the internally generated events. Naydich and Nowakowski [6] have explored this option in detail for the CoRE FGS specification. They translated the specification into PROMELA, and used the SPIN model checker to understand the consequences of using a micro-time semantics for this specification.

4.6 Miscellaneous Comments

Here we note a few other points in the CoRE specification that deserve comment.

4.6.1 `mode_Autopilot` Entering `ENGAGED`

The `mode_Autopilot` mode machine has a mode `ENGAGED` with two submodes `Normal` and `Sync`. The `ENGAGED` submode transition table and the descriptive text suggest that the authors' intent is that `mode_Autopilot` be in `ENGAGED/Sync` whenever `term_SYNC = TRUE`. However, `mode_Autopilot = ENGAGED` is initialized to the `Normal` submode, and there is no statement that `mode_Autopilot= ENGAGED` should initialize to the `SYNC` submode if `term_SYNC = TRUE` when `mode_Autopilot` enters `ENGAGED`.

4.6.2 `mon_Nav_Source_Signal_Type`

There is no explicit invariant given between `mon_Nav_Source_Signal_Type<VNR<`N`>>` and `mon_Nav_Source_Frequency<VNR<`N`>>`. Without such an invariant, it is possible for the value of `mon_Nav_Source_Signal_Type(mon_Selected_Nav_Source)` to change from, say, `LOC` to `VOR` without causing the event `@Nav_Source_Change`; this would also cause the variable `term_Selected_Nav_Type` to change. Thus Invariant 7, which asserts that `mode_Active_Lateral = APPR/Track` $\Rightarrow$ `term_Selected_Nav_Type` $\in$ {`LOC, FMS`} could be violated. Apparently, the intent is that `mon_Nav_Source_Signal_Type<VNR<`N`>>` is supposed to be a function of `mon_Nav_Source_Frequency<VNR<`N`>>`, and cannot change independently. However, this is not stated explicitly in the CoRE specification.

4.6.3 `Duration(INMODE)` Booleans

Transitions 43 and 53, both in the `mode_Active_Vertical` transition table, may not be disjoint. Transition 43 occurs exactly when Transition 64 occurs, and Transition 53 occurs exactly when Transition 69 occurs. Here are the transitions in question.

`mode_Altitude_Select ENABLED` Submode Transition Table

Id	From	Event	To
64	`ARMED`	`@T(term_ALTSEL_Cond = Capture AND Duration(INMODE) > const_min_armed_period)`	`ACTIVE`

`mode_Vertical_Approach ENGAGED` Transition Table

Id	From	Event	To
69	`ARMED`	`@T(term_Vertical_Approach_Cond_Met AND Duration(INMODE) > const_min_armed_period)`	`TRACK`

The transition of `mode_Altitude_Select` into `ACTIVE` triggers Transition 43, and the transition of `mode_Vertical_Approach` to `TRACK` triggers Transition 53.

Now, it is not clear whether a `Duration(INMODE)` Boolean (which is not really an input event) can become `TRUE` at the same time that, say, `term_Vertical_Approach_Track_Cond_Met` becomes `TRUE`. If these two can become `TRUE` simultaneously, then the two transitions above could happen simultaneously.

Furthermore, if it is possible that `mode_Altitude_Select` enters `ARMED` at the same time that `mode_Vertical_Approach` enters `ARMED`, then both of their `Duration(INMODE)` Booleans could become `TRUE` simultaneously. Then if both `term_ALTSEL_Cond = Capture` and `term_Vertical_-Approach_Cond_Met = TRUE` before the `Duration(INMODE)` Booleans become `TRUE`, then the two transitions could go off.

These two scenarios could be prevented by ensuring that `term_ALTSEL_Cond = Capture` and `term_Vertical_Approach_Cond_Met = TRUE` are mutually exclusive. While this is probably the authors' intent, it is not stated in the specification.

Chapter 5

Preliminaries on the Z Notation

5.1 Introduction

Z (pronounced *zed*) is a formal specification language [4, 10, 11]. Thus it has a strictly defined syntax and semantics, and it is very useful in describing what a piece of software is supposed to do. However, it is by its nature not an executable language, as it describes only the objective of the program, not how to accomplish it.

The Z language is well-suited to handling an event-driven system with finitely many states. Z deals in variables and predicates, which are expressions of the variables that are either true or false. All variables in Z are typed. Examples of types are the natural numbers, or a free type with enumerated elements (such as {HDG, GA, ...}). The free type construction in Z allows us to, for instance, enumerate the possible modes of a mode machine.

5.2 Schemas

Z is essentially first-order predicate calculus plus schemas. A schema models the state of a system; so we can use schemas to model change, which is essential for specifying state machines. For instance,

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{A\_Schema}} \\
\hline
\textit{Var} : \mathbb{N} \\
\hline
1 \leq \textit{Var} \leq 50 \\
\hline
\end{array}$$

declares a variable *Var* with values in the naturals, and constrains Var to have values between 1 and 50.

We can build more complicated schemas by including this one in the declaration part of another, such as below:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Another\_Schema}} \\
\hline
\textit{A\_Schema} \\
\textit{Var2} : \mathbb{N} \\
\textit{Var3} : \mathbb{N} \\
\hline
\textit{Var2} = \textit{Var} + 5 \\
\textit{Var3} \geq 0 \\
\hline
\end{array}$$

This schema declares two new variables *Var2* and *Var3*, and then asserts an invariant relating *Var2* and *Var*, and another involving *Var3*. A Z convention is that predicates stated on separate lines are conjoined.

We illustrate these concepts with a sample mode machine. The actual Z specification in the appendix is constructed along these lines.

We can define a free type for the modes of *mode_Active_Lateral* as follows.

$$LATERAL\_MODE ::= L\_GA \mid L\_APPR \mid HDG \mid NAV \mid ROLL$$

We can define the *mode_Active_Lateral* mode machine as a variable whose values are in the *LATERAL_MODE* free type.

$$\begin{array}{|l}
\text{\_\_} Def\_Of\_mode\_Active\_Lateral \text{\_\_\_\_\_\_\_\_\_\_} \\
mode\_Active\_Lateral : LATERAL\_MODE \\
\hline
\end{array}$$

Once we have defined several mode machines, we can combine them together into one large schema.

$$\begin{array}{|l}
\text{\_\_} Mode\_Machines \text{\_\_\_\_\_\_\_\_\_\_} \\
Def\_Of\_mode\_Active\_Lateral \\
Def\_Of\_mode\_Active\_Vertical \\
Def\_Of\_mode\_Altitude\_Select \\
Def\_Of\_mode\_Vertical\_Approach \\
\hline
\end{array}$$

We can declare our other variables in the same way.

$$\begin{array}{|l}
\text{\_\_} Some\_Variables \text{\_\_\_\_\_\_\_\_\_\_} \\
Var1 : \mathbb{N} \\
Var2 : VAR2 \\
\ldots \\
\hline
\end{array}$$

$$\begin{array}{|l}
\text{\_\_} More\_Variables \text{\_\_\_\_\_\_\_\_\_\_} \\
Var3 : \mathbb{N} \\
Var4 : VAR4 \\
\ldots \\
\hline
\end{array}$$

Then we can assemble everything into a large State schema.

$$\begin{array}{|l}
\text{\_\_} State \text{\_\_\_\_\_\_\_\_\_\_} \\
Mode\_Machines \\
Some\_Variables \\
More\_Variables \\
\hline
\end{array}$$

Once we have defined a schema, we can use a powerful convention of Z that allows us to declare another type of variable, namely a *schema binding.* Any schema can be viewed as the set of all values of the variables of the schema that also satisfy the predicate of the schema. Thus the schema *State* can be viewed as the set of all possible states of the system. An arbitrary element of this set, called a *schema binding*, can be referred to by the shorthand $\theta State$ (reminiscent of "the" *State*).

5.3 Transitions

The Z notation provides a convention for transitions, using *operation schemas*. Given a schema named *State*, for instance, we can define another copy $State'$ of the schema, all of whose variables have the same names as the variables of *State*, except that they are primed. The convention is that the unprimed State represents the "before" state, and the primed $State'$ represents the "after". Then the schema Δ *State*, by convention, contains both *State* and $State'$, and is considered the "change in State".

Thus, to specify a transition in Z, it suffices to write down a schema that includes Δ *State* and asserts a relation between the values of *State* and the values of $State'$. The Z convention is that a transition is simply a constraint between the old and new states, such as

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{A\_Transition}} \\
\Delta State \\
\hline
Mode = A \land Event1 \land Mode' = B \\
\hline
\end{array}$$

Note that the Z operation schema does not mention implication; the predicate is not $Mode = A \land Event1 \Rightarrow Mode' = B$. This is because it is not meant to always hold as a predicate on $\Delta State$. Rather, it is meant to be one of several transitions, and the disjunction of these transitions is to be the total operation. See Section 6.8.3 for an example.

5.4 Axiomatic Definitions

Another important feature of Z is the use of axiomatic definitions. These are used to globally define constants, functions, or subsets. In contrast to a schema, where each binding of a schema produces a separate copy of the variables inside it, an axiomatic definition produces one globally accessible definition of a variable.

For instance, the constant *const_min_armed_period* would be declared in an axiomatic definition as follows:

$$\begin{array}{|l}
const\_min\_armed\_period : \mathbb{N} \\
\hline
const\_min\_armed\_period = 500
\end{array}$$

5.5 Booleans in Z

One aspect of Z which may seem disconcerting at first is the lack of a built-in Boolean data type. In Z, all relations (and thus functions) are defined as sets; a function is simply a special kind of binary relation, and is thus a set of ordered pairs. Thus, for example, the unary relation *odd(x)* is in fact a subset of *Z* consisting of all the odd integers. It is not natural in Z to define a function *odd(x)* that takes an integer x to a value "*TRUE*" if x is odd and "*FALSE*" otherwise. In fact, *TRUE* and *FALSE* are predicates in Z; and in Z, predicates are not merely "expressions with Boolean values", as they are in many other languages.

This lack of Boolean types is in concordance with Z's philosophy of specifications. Since *TRUE* and *FALSE* has no built-in advantage over, say, *VALID* and *INVALID*, we are encouraged to use appropriate names for the values of the variables.

In our treatment of the FGS, we have defined a Boolean type TRU and $FALS$ and used it rather often, in order to facilitate comparison with the CoRE specification. The reason for changing the spelling is that Z/EVES 1.4 may mistakenly print out the predicate *(true)* instead of $TRUE$.

Chapter 6

Formal Specification of the FGS

6.1 Translating the Specification into Z

We have chosen to organize the Z specification in a similar manner to the CoRE specification. The definitions are roughly in the same order as they appear in the CoRE specification. Z requires all variables to be declared before they can be otherwise used, so we occasionally had to reorganize some of the variable declarations. The variables tend to have the same names unless they did not appear in the CoRE specification at all. Some mode names, such as `GA` and `APPR`, were overloaded in the CoRE specification; they stood for two mode values, one for `mode_Active_Lateral` and one for `mode_Active_Vertical`. Since Z requires that different free types (such as the possible modes of *mode_Active_Lateral* and *mode_Active_Vertical*) be disjoint, we changed some names slightly. For instance, we use *L_APPR* and *V_APPR* instead of just `APPR`.

Many of the difficulties that we encountered in translating the CoRE specification into Z arise because the CoRE model is purely event-driven: an event occurs and its effect is felt instantaneously. Therefore, when we translate the FGS into a polling cycle model, we must decide on interpretations of certain aspects of the FGS accordingly.

We have chosen to interpret the CoRE FGS specification using, as much as possible, the discrete-time formal model for SCR semantics [2]. In particular, we use its definitions of events and simultaneity of events. The SCR model is quite compatible with Z, since events and transitions are defined in terms of comparisons between two system states (unprimed and primed), just as in Z. However, in order to analyze the CoRE specification, we must ignore some constraints of the SCR model (in particular, the requirement that the variables be partially ordered by dependency). In doing so, we give up some of the benefits, such as the sufficiency of static consistency checks in determining exhaustiveness and determinism of the system.

For instance, consider the example discussed in Section 4.2. The most straightforward translation of Transitions 28 and 57 into a discrete-time model is as follows. Transition 28 would translate to

$$
\begin{array}{|l}
\hline
\textit{Transition\_TwentyEight} \\
\textit{Transition} \\
\hline
\textit{mode\_Active\_Vertical} = \textit{V\_GA} \\
\textit{mode\_Active\_Vertical}' \neq \textit{V\_GA} \\
\textit{mode\_Active\_Lateral} = \textit{L\_GA} \\
\textit{mode\_Active\_Lateral}' \in \textit{ROLL} \\
\hline
\end{array}
$$

and Transition 57 would translate into

$Transition\_FiftySeven$

$Transition$

$$
\begin{aligned}
&mode\_Active\_Lateral = L\_GA \\
&mode\_Active\_Lateral' \neq L\_GA \\
&mode\_Active\_Vertical = V\_GA \\
&mode\_Active\_Vertical' = PITCH
\end{aligned}
$$

Clearly these two transitions will conflict with any transition that takes these two modes from L_GA and V_GA to any modes other than $PITCH$ and $ROLL$.

Furthermore, these two transitions, by themselves, are nondeterministic. If $mode\_Active\_Lateral = L\_GA \wedge mode\_Active\_Vertical = V\_GA$, then the above two predicates are satisfied if $mode\_Active\_Lateral' = L\_GA \wedge mode\_Active\_Vertical' = V\_GA$, but they are equally satisfied if $mode\_Active\_Lateral' \in ROLL \wedge mode\_Active\_Vertical' = PITCH$. Thus we cannot exclude the possibility of "spontaneous transition" if both transitions are allowed to hold simultaneously.

If Transitions 28 and 57 were the only transitions in their respective tables, then the tables would individually be nondisjoint, but the system as a whole would still be nondeterministic.

6.2 Initializing the Variables

The CoRE specification requires that each variable be initialized to a certain value. However, many initial values are computed from the other variables' initial values using condition or event tables.

In Z, initialization is described as follows. Suppose our schema that declares the variables is called *State*. Then we define another schema, conventionally called *InitState*, whose predicates contain the initialization of the variables.

For example, suppose we have a schema

$State$

$$
\begin{aligned}
&Var1 : 1 \,..\, 50 \\
&Var2 : 1 \,..\, 600
\end{aligned}
$$

Then a possible initialization would be

$InitState$

$State$

$$
\begin{aligned}
&Var1 = 25 \\
&Var2 = 1
\end{aligned}
$$

The *InitState* schema is merely a constraint on the possible initializations. If not all variables are given a value in *InitState*, then there could be several possible initializations that satisfy *InitState*.

Of course, it is imperative that the initialization actually be a legal state. The legality check for the initial state takes the form of a theorem asserting that there in fact exists a state (i.e. a binding of *State*) that satisfies the predicates of the schema *InitState*; that is, the values assigned in *InitState* are not inconsistent with the predicates given in State. The spot is the Z "such that" notation.

Theorem Init_Is_OK:
$\exists \mathit{State} \bullet \mathit{InitState}$

The CoRE specification is not very explicit about when the FGS is to be initialized. Is it to be activated when the plane is still on the ground? Is it possible that the FGS will turn on when the plane is in the air and is already overspeed? Should this cause the `mode_Overspeed` machine to initialized to `TOO_FAST`?

6.3 Numerical Quantities in Z

Z does not possess a built-in type for real numbers. Its only built-in type is the integers, along with some subsets such as the naturals. The CoRE specification is not very explicit about some of the precisions to which numerical values should be stored (such as `con_Selected_Heading_Annunciation`, which is in degrees, but the number of significant figures is not declared).

In Z, we can declare a type, say *ALTITUDE*, as a copy of the the naturals, and then we can declare variables whose values are of type *ALTITUDE*. However, Z/EVES does not perform this kind of typechecking; if both *ALTITUDE* and *AIRSPEED*, say, are copies of the naturals, Z/EVES will not distinguish between them.

In order to translate the numerical types of the CoRE specification into Z, we have chosen a precision that is consistent with the CoRE declarations. For instance, if a CoRE variable declaration is given as -5.0 to 5.0 degrees, then our corresponding Z declaration would be -50 ...50. In the CoRE specification, several variables of type `Altitude_Rate` were measured to different precisions. We chose to declare the Z type *ALTITUDE_RATE* with the largest increments (0.001 kft/min) that all allowed all of these variables to be measured to the desired precision with an integral number of increments.

6.4 Transitions

We are operating under the assumption that only one monitored variable changes during each polling cycle. Therefore, we want to tag each cycle with the input event that occurs in that cycle. First we must define a free type *EVENT* containing all of the event labels that we will use.

$\mathit{EVENT} ::= \mathit{At\_Switch\_Pressed} \mid \mathit{Event2} \mid \mathit{No\_Event} \ldots$

Then we declare a variable taking values in EVENT.

The_Event
$\mathit{event} : \mathit{EVENT}$

Since each event constitutes a change in a monitored variable, we define an event using a schema of the following form. Suppose we have a switch *Switch* taking the values *ON* and *OFF*, declared as follows.

$\mathit{SWITCH} ::= \mathit{ON} \mid \mathit{OFF}$

A_Switch
$\mathit{Switch} : \mathit{SWITCH}$

Then we can define the event *At_Switch_Pressed* corresponding to the *Switch* going from *OFF* to *ON* as follows

$$\begin{array}{|l}
\textit{Event\_Switch\_Pressed} \\
\hline
\Delta A\_Switch \\
The\_Event \\
\hline
(event = At\_Switch\_Pressed) \Leftrightarrow (Switch = OFF \wedge Switch' = ON) \\
\hline
\end{array}$$

For convenience, we have overloaded some of the event labels. For instance, sometimes there are two switches with the same function; in this case, pushing either will generate the same event. There are also some other places where the overloading is a bit greater, such as *ALTSEL_Track_Cond_Met*. In this case, the variable *term_ALTSEL_Track_Cond* is a compound Boolean, and it would be rather inconvenient to list the changes of all of the monitored variables as separate input events.

Some CoRE events are defined essentially as disjunctions of other events; for instance, `@Lateral_Mode_Requested` occurs exactly when one of `@HDG_Switch_Pressed`, `@NAV_Switch_Pressed`, `APPR_Switch_Pressed`, or `GA_Pressed` occurs. to simulate this in Z, we define a subset *Lateral_Mode_Requested* consisting of the appropriate input events. Then we translate the event `@Lateral_Mode_Requested` into the statement *event* $\in$ *Lateral_Mode_Requested*.

We must be conservative about overloading, because it is easy to get confused about what should be considered an input event and what is properly an internal event. The labeling is designed specifically to guarantee the One Input Assumption; so we must be sure that an external event can influence only one event label.

6.5 Sustaining Conditions

CoRE allows parts of the specification, such as variables, to be "tagged" with sustaining conditions. A sustaining condition is a predicate, and the tagged part is not to be accessed or changed in any way unless the predicate is true. So, for instance, a variable with a sustaining condition is not supposed to be accessed or changed unless the sustaining condition is true.

However, the CoRE Guidebook [1] is not very specific on what variables can or cannot be tagged, and it does not give much idea of the best way to translate such a condition into our polling-cycle model. For instance, is it permissible to access or to change a tagged variable in the same polling cycle that the sustaining condition changes?

As well, the CoRE FGS specification tags other quantities such as mode machines. Since a mode machine records state history, the mode of a mode machine cannot only be calculated from the values of variables in the present state. The CoRE FGS specification even tags invariants with sustaining conditions; since some variables involved have sustaining conditions, the authors must restrict the validity of the invariant to the states where such variables can be accessed.

A further indication of the authors' intent is given by the use of the sustaining condition `mode_Flight_Director = ON` for the mode machines `mode_Active_Lateral, mode_Active_Vertical, mode_Altitude_Select, mode_Vertical_Approach`. For instance, suppose `mode_Flight_Director = OFF`, so that these four mode machines are inaccessible. Now suppose that a flight mode is requested: say `@HDG_Switch_Pressed` occurs. First, note that the this event causes `mode_Flight_Director′ = ON`. It is clear that the `mode_Active_Lateral` table is in effect for that polling cycle, so that `mode_Active_Lateral′ = HDG`.

So a sustaining condition is not quite like a guard in a conditioned ("when") event. A variable in a given state can be accessed as long as the sustaining condition is true in that same state. For term variables, our interpretation is that the tagged variable can be changed only when the sustaining condition is `TRUE` in the new state.

However, in the case of the mode machines, it seems important that we use the mode machine's value to keep track of the sustaining conditions. To each mode machine, we add an *NOT_IN_MODE* mode value, and add transitions to ensure that the mode machine is in *NOT_IN_MODE* exactly when it sustaining condition is not true. So it seems that a reasonable interpretation is to state an invariant of the system, that each mode machine is in its corresponding *NOT_IN_MODE* mode whenever its sustaining condition is not true. Furthermore, in the transition tables, we add two transitions detailing what happens when the sustaining condition becomes true, and when the sustaining condition becomes false. In the case when an invariant has a sustaining condition, we have rewritten the invariant to say that the sustaining condition implies the invariant.

6.6 Hierarchical Mode Machines

The authors of the CoRE FGS specification extended the CoRE method by allowing the use of *hierarchical mode machines*, where a given mode of a mode machine may have its own submodes, which can be governed by a separate transition table.

For instance, the `Mode_Altitude_Select` mode machine has two "top-level" modes, `CLEARED` and `ENABLED`. The `ENABLED` submode has two submodes, `ARMED` and `ACTIVE`. The `ACTIVE` submode has two submodes, `Capture` and `Track`.

One way to model the `mode_Altitude_Select` machine in Z would be to construct separate, current mode machines for each submode. However, this requires two more mode machines (say `ALTSEL_ENABLED_Mode` and `ALTSEL_ACTIVE_Mode`). Since all Z variables have to have a value at all times, we would be wise to construct extra `NOT_IN_MODE` values for these machines, for the times when the the top-level mode machine is not in the respective mode.

A less complicated alternative is to flatten the hierarchical mode machine, and then to introduce subsets corresponding to the top-level modes. For instance, we could define the free type for *mode_Altitude_Select* as follows:

$$\begin{aligned} ALTSEL\_MODE ::= {} & ALTSEL\_CLEARED \mid ALTSEL\_ARMED \mid ALTSEL\_CAPTURE \\ \mid {} & ALTSEL\_TRACK \end{aligned}$$

Then we give an axiomatic definition

$$\begin{array}{|l} ALTSEL\_ENABLED, ALTSEL\_ACTIVE : \mathbb{P}\, ALTSEL\_MODE \\ \hline ALTSEL\_ENABLED = \{ALTSEL\_ARMED, ALTSEL\_CAPTURE, ALTSEL\_TRACK\} \\ ALTSEL\_ACTIVE = \{ALTSEL\_CAPTURE, ALTSEL\_TRACK\} \end{array}$$

Then any CoRE reference to `mode_Altitude_Select = ACTIVE` would be replaced in Z with `mode_-Altitude_Select ∈ ALTSEL_ACTIVE`.

Most CoRE mode machines are given with an initial value. For instance, the CoRE `ACTIVE` submode machine has the initial value `Capture`. So we can translate a CoRE transition that sends `mode_Altitude_Select` to `ACTIVE` into a Z transition that sends *mode_Altitude_Select* to *ALTSEL_CAPTURE*.

However, some submode machines are given with transition tables that discuss entry into the mode. In this case, the transitions that give entry into the mode should be broken up into different transitions, one for each possible submode.

This approach seems compatible with the intent of the authors of the CoRE specification. In particular, this approach also allows us to capture the intent of the `ENABLED` supermodes in the `mode_Altitude_Select` and `mode_Vertical_Approach` mode machines. In the CoRE specification, one is allowed to write a transition that sends `mode_Altitude_Select` to `ENABLED`; but rather the submode values `ARMED` or `ACTIVE` are actually the values exported to other mode machines. In Z we translate a transition into `ENABLED` as a predicate asserting that $mode\_Altitude\_Select'$ lies in the subset $ENABLED$.

6.7 INMODE Booleans

Several places in the CoRE specification use a Boolean such as `@T(Duration(INMODE) > 10 sec)`, as in the following table.

`mode_Autopilot` DISENGAGED Transition Table			
Id	From	Event	To
11	`Warning`	`@T(Duration(INMODE(Warning)) > 10 sec)`	Normal

In our Z translation, we construct a Boolean for each mode and duration that is used in the specification. For the above, we declare

$$\begin{array}{|l}
\multicolumn{1}{l}{INMODE\_Boolean} \\
\hline
Duration\_INMODE\_AP\_Disengaged\_Warning\_gt\_ten\_sec : BOOLEAN \\
\hline
\end{array}$$

which is to be TRU when $mode\_Autopilot$ has been in the mode $AP\_DISENGAGED\_WARNING$ for more than 10 seconds. (We do not, however, specify a clock or a mechanism for changing the value of this Boolean; that can be done at a later stage in the design process.)

We make one constraint, which is that if this Boolean (which we abbreviate to $Duration\_INMODE$ for now) is TRU, then the corresponding mode machine was in the appropriate state during the previous polling cycle.

$$\begin{array}{|l}
\multicolumn{1}{l}{Transition\_INMODE\_Requirement} \\
\hline
\Delta INMODE\_Boolean \\
\Delta State \\
\hline
Duration\_INMODE\_AP\_Disengaged\_Warning\_gt\_ten\_sec' = TRU \\
\Rightarrow (mode\_Autopilot = DISENGAGED\_WARNING) \\
\hline
\end{array}$$

This constraint might seem odd, since it would seem reasonable to require if the $Duration\_INMODE$ Boolean is TRU in a given state, then the corresponding mode machine is that mode. However, to capture the intent of Transition 11 above, where $Duration\_INMODE = FALS \wedge Duration\_INMODE' = TRU$ is used as a trigger for setting $mode\_Autopilot' = AP\_DISENGAGED\_NORMAL$, it seems most reasonable to make our constraint as above.

We have not been more specific with the details of these Booleans, such as specifying a clock. This is concordant with the use of Z; we can add a more specific constraint at a later point in the specification lifecycle.

6.8 Translating Tables

A CoRE table associated to a variable assigns a value to that variable. A condition table for a variable decides, based on the values of certain other variables, which of several values to assign the variable. An event table or mode transition table decides, based on the changes in certain other variables that occur in a given transition, which of several values to assign to the variable in the new state.

6.8.1 Condition Tables

The translation of a CoRE condition table to our Z polling cycle model is straightforward.

Suppose our condition table is

Sample Condition Table

`Mode = A`	`Var1 = T`	`Var1 = F`	`X`
`Mode = B`	`X`	`X`	`TRUE`
`CondVar`	`Val1`	`Val2`	`Val3`

Given the schema `State`, we write out the conditions of the table, first by rows, and then within each row, by columns. We suppose that the variables that are involved in the condition table are declared within the schema *Variables*.

Condition_Table

Variables

$$(Mode = A \land Var1 = T \land CondVar = Val1) \lor$$
$$(Mode = A \land Var1 = F \land CondVar = Val2) \lor$$
$$(Mode = B \land CondVar = Val3)$$

Since the condition table for a variable defines that variable in terms of other variables in the same state, we shall include condition tables as part of the *State* schema. Then in each state, the condition table will hold and will thus compute the value of its variable.

6.8.2 Event Tables

An ordinary event table also has a straightforward translation. Suppose we have an event table of the following form.

Sample Event Table

`Mode1 = A`	`Event1`	`Event2`	`X`
`Mode1 = B`	`Event2`	`Event1`	`Event3`
`EventVar`	`Val1`	`Val2`	`Val3`

We can translate the table into Z as follows. We suppose that the variables involved in the event table are declared in *Variables*, and the input event variable is declared in *The_Event*.

Event_Table

$\Delta \mathit{Variables}$
$\mathit{The\_Event}$

$$
\begin{array}{l}
(\mathit{Mode1} = A \wedge \mathit{Event1} \wedge \mathit{EventVar}' = \mathit{Val1}) \vee \\
(\mathit{Mode1} = A \wedge \mathit{Event2} \wedge \mathit{EventVar}' = \mathit{Val2}) \vee \\
(\mathit{Mode1} = B \wedge \mathit{Event2} \wedge \mathit{EventVar}' = \mathit{Val1}) \vee \\
(\mathit{Mode1} = B \wedge \mathit{Event1} \wedge \mathit{EventVar}' = \mathit{Val2}) \vee \\
(\mathit{Mode1} = B \wedge \mathit{Event3} \wedge \mathit{EventVar}' = \mathit{Val3}) \vee \\
(\neg\, ((\mathit{Mode1} = A \wedge \mathit{Event1}) \vee (\mathit{Mode1} = A \wedge \mathit{Event2}) \vee \\
(\mathit{Mode1} = B \wedge \mathit{Event2}) \vee (\mathit{Mode1} = B \wedge \mathit{Event1}) \vee \\
(\mathit{Mode1} = B \wedge \mathit{Event3})) \wedge \mathit{EventVar}' = \mathit{EventVar})
\end{array}
$$

The negation of the disjunction of all the event triggers is used as the trigger for no change in the value of *EventVar*. This construction is necessary in order to define the value of *EventVar* for all possible changes of *State*.

One type of event used in the CoRE specification that deserves special mention is the `ENTERED` (otherwise known as `@T(Inmode)`) event. This event occurs when the mode machine enters the mode corresponding to the row in which the `ENTERED` appears. Of course, this means that the mode machine was not in that mode in the initial state, but was in that mode in the final state. The SCR method arranges the event table so that each row corresponds to a possible old mode of the system, and events listed in that row may happen while the old mode is equal to that row's mode value. The translation of the hypothesis for an `ENTERED` event is still straightforward, but the old state will not be in the mode where the `ENTERED` event is listed, so we should not compare this event to the other events in that row.

In order to partition the events properly, we must rewrite the `ENTERED` events. If an `ENTERED` event appears in the i-th row corresponding to the mode value `M_i`, then we rewrite the event as `@T(Mode = M_i)` and transfer this event to all of the other rows (since those correspond to the possible old values of `Mode` before `@T(Mode = M_i)`. For instance, suppose we have a table as below.

Sample Event Table with `@T(INMODE)`			
`Mode1 = A`	`@T(INMODE)`	`Event2`	`X`
`Mode1 = B`	`X`	`Event1`	`Event3`
`Mode1 = C`	`X`	`Event1`	`Event2`
`EventVar`	`Val1`	`Val2`	`Val3`

The `@T(INMODE)` really means `@T(Mode1 = A)`. So we should remove this event from the row `Mode1 = A` and put it in the other rows where it could occur, namely the rows for `Mode1 = B` and `Mode1 = C`.

The rewritten table would be

Rewritten Event Table with `@T(INMODE)`			
`Mode1 = A`	`X`	`Event2`	`X`
`Mode1 = B`	`@T(Mode1 = A)`	`Event1`	`Event3`
`Mode1 = C`	`@T(Mode1 = A)`	`Event1`	`Event2`
`EventVar`	`Val1`	`Val2`	`Val3`

We also must discuss the translation of the `CONTINUOUSLY` events. For instance, let us look at one row of a table containing such an event.

Continuous Event Table for `EventVar`			
`Mode = A`	`CONTINUOUSLY WHEN Bool = TRUE`	`Event2`	
`EventVar`	`Val1`	`Val2`	

Since this is an event table, we have decided that the polling cycle translation should only specify the value of $EventVar'$ in the new state, rather than stating an invariant about the unprimed $EventVar$ as well. We believe that the authors' intent is that $EventVar$ should equal $Var1$ in any state where $Bool = TRU$ and $Mode = A$. So we make this assertion for the new (primed) state, and then translate the other entries of the table in order not to conflict with this assertion.

Continous_Event_Table

$\Delta State$

$(Mode' = A \land Bool' = TRU \land EventVar' = Val1) \lor$
$(Mode = A \land Bool' = FALS \land Event2 \land EventVar' = Val2)$

Now let us demonstrate on a table that contains both `ENTERED` and `CONTINUOUS` events. We abbreviate `mode_Flight_Director` to **FD** and `mode_Active_Vertical` to **AV**. Here, because two mode machines are involved, the `ENTERED` could mean either **AV** entering `FLC` while **FD** is `ON`, or also **FD** entering `ON` at the same time as **AV** enters `FLC`. We believe that the authors' intent is that the both of these events will trigger the transition in question.

FD	**AV**	Event	
`OFF`	N/A	`X`	`X`
`ON`	$\overline{\texttt{FLC}}$	`X`	`@Speed_Knob_Changed`
`ON`	`FLC`	`ENTERED`	`@Speed_Knob_Changed`
		`CONTINUOUSLY WHEN term_SYNC`	
`term_Reference_IAS`		`mon_Indicated_Airspeed`	`limit(0, 512, term_Reference_IAS' + 1*(term_Speed_ Knob_Rotation))`

Some of this translation, such as using $\in$ instead of $=$ in some places, comes from other aspects of the translation into Z.

$$
\begin{array}{l}
\textbf{Schema } \mathit{Def\_Of\_term\_Reference\_IAS} \\
\hline
\Delta \mathit{aggr\_References} \\
\Delta \mathit{aggr\_FCP\_Knobs} \\
\Delta \mathit{aggr\_Air\_Data} \\
\Delta \mathit{SYNC} \\
\mathit{The\_Event} \\
\Delta \mathit{aggr\_Flight\_Modes} \\
\hline
(\mathit{mode\_Flight\_Director}' \in \mathit{FD\_ON} \land \mathit{mode\_Active\_Vertical} \notin \mathit{FLC} \land \\
\mathit{mode\_Active\_Vertical}' \in \mathit{FLC} \land \mathit{term\_SYNC}' = \mathit{FALS} \\
\land\ \mathit{term\_Reference\_IAS}' = \mathit{mon\_Indicated\_Airspeed}') \\
\lor\ (\mathit{mode\_Flight\_Director}' \in \mathit{FD\_ON} \\
\land\ \mathit{mode\_Active\_Vertical}' \in \mathit{FLC} \land \mathit{term\_SYNC}' = \mathit{TRU} \\
\land\ \mathit{term\_Reference\_IAS}' = \mathit{mon\_Indicated\_Airspeed}') \\
\lor\ (\mathit{mode\_Flight\_Director} \in \mathit{FD\_ON} \land \mathit{mode\_Active\_Vertical} \notin \mathit{FLC} \\
\land\ \mathit{event} = \mathit{Speed\_Knob\_Changed} \\
\land\ \mathit{mon\_Indicated\_Airspeed}' = \mathit{term\_Reference\_IAS}' = \\
\mathit{min}\{512, \mathit{max}\{0, \mathit{term\_Reference\_IAS} + \mathit{term\_Speed\_Knob\_Rotation}'\}\}) \\
\lor\ (\mathit{mode\_Flight\_Director} \in \mathit{FD\_ON} \land \mathit{mode\_Active\_Vertical} \in \mathit{FLC} \\
\land\ \mathit{term\_SYNC}' = \mathit{FALS} \land \mathit{event} = \mathit{Speed\_Knob\_Changed} \\
\land\ \mathit{mon\_Indicated\_Airspeed}' = \mathit{term\_Reference\_IAS}' = \\
\mathit{min}\{512, \mathit{max}\{0, \mathit{term\_Reference\_IAS} + \mathit{term\_Speed\_Knob\_Rotation}'\}\}) \\
\lor\ (\neg\ ((\mathit{mode\_Flight\_Director}' \in \mathit{FD\_ON} \land \mathit{mode\_Active\_Vertical} \notin \mathit{FLC} \land \\
\mathit{mode\_Active\_Vertical}' \in \mathit{FLC} \land \mathit{term\_SYNC}' = \mathit{FALS}) \\
\lor\ (\mathit{mode\_Flight\_Director}' \in \mathit{FD\_ON} \\
\land\ \mathit{mode\_Active\_Vertical}' \in \mathit{FLC} \land \mathit{term\_SYNC}' = \mathit{TRU}) \\
\lor\ (\mathit{mode\_Flight\_Director} \in \mathit{FD\_ON} \land \mathit{mode\_Active\_Vertical} \notin \mathit{FLC} \\
\land\ \mathit{event} = \mathit{Speed\_Knob\_Changed}) \\
\lor\ (\mathit{mode\_Flight\_Director} \in \mathit{FD\_ON} \land \mathit{mode\_Active\_Vertical} \in \mathit{FLC} \\
\land\ \mathit{term\_SYNC}' = \mathit{FALS} \land \mathit{event} = \mathit{Speed\_Knob\_Changed})) \\
\land\ \mathit{term\_Reference\_IAS}' = \mathit{term\_Reference\_IAS}) \\
\hline
\end{array}
$$

6.8.3 Transition Tables

We can translate the the transition tables of the CoRE specification into Z transition schemas. For instance, consider a table for a certain mode machine `M` with modes `A`, `B`, `C`, . . . and transition table with events Event1 and Event2, that looks like:

Sample Transition Table for `Mode`		
From	Event	To
`A`	`Event1`	`B`
`B`	`Event2`	`C`
`C`	`Event1`	`A`

Recall the structure of an operation schema from Section 5.3. We translate the rows of the transition table into schemas as follows. We presume that all of the variables in the table appear in the *Variables* schema.

$Transition\_One$

$\Delta Variables$

$Mode = A \land Event1 \land Mode' = B$

$Transition\_Two$

$\Delta Variables$

$Mode = B \land Event2 \land Mode' = C$

$Transition\_Three$

$\Delta Variables$

$Mode = C \land Event1 \land Mode' = A$

We must also specify that that value of *Mode* does not change when none the above events occurs.

$Transition\_Four$

$\Delta Variables(\neg\,((Mode = A \land Event1) \lor (Mode = B \land Event2) \lor (Mode = C \land Event1))$
$\land\ Mode' = Mode)$

Finally, the translation of the transition table is the disjunction of the above four schemas. This schema thus asserts that at least one of the above operations must hold.

$$\begin{array}{l} Transition\_Table \mathrel{\widehat{=}} \\ Transition\_One \lor \\ Transition\_Two \lor \\ Transition\_Three \lor \\ Transition\_Four \end{array}$$

As we noted before, this translation has the property that if two rows are not disjoint, then the table is nondeterministic; either outcome will satisfy the predicate of the table. This is in contrast to an imperative translation of the table, with, a conjunction of predicates like $Mode = A \land Event1 \Rightarrow Mode' = B$; such a conjunction would be inconsistent and thus equivalent to *false* if two rows were in conflict.

6.9 Formal Verification of Properties of the FGS

We can use our Z specification to construct formal verifications of desirable properties of the specification. For instance, for each condition, event, or transition table, we can generate a theorem to check disjointness of the entries; and for condition tables, we generate a theorem to check that the table covers all possibilities.

In addition, we may have additional *invariants* about the specification that we wish to prove. For instance, the CoRE FGS specification requires that `mode_Active_Vertical` ∈ {`APPR, GA, ALTHOLD`} ⇔ `mode_Altitude_Select = CLEARED`. A usual method of showing that an invariant holds for all states is to show that it holds for one state of the system (such as the initial state), and then to show that every transition preserves the invariant.

We are using Z/EVES 1.4 [7, 8], a theorem prover for the Z language, to generate proofs of table checks and system invariants.

- Z/EVES can perform syntax and type checking of a Z specification.
- Z/EVES can perform schema expansion, which replaces an included schema by its text; this is very useful, especially in a large specification.
- Z/EVES can do precondition calculation, to determine the necessary conditions for an operation to be invoked; not taking into account all of the preconditions of an operation is a frequent cause of actual program failure.
- Z/EVES can perform domain checking, to ensure that functions are applied on elements that are actually on their domain.
- Z/EVES provides an interface to the EVES theorem prover, which provides powerful automated support (e.g. heuristics and conditional rewriting) as well as user commands to direct the theorem prover.

6.10 General Principles

The EVES theorem prover is designed to manipulate predicates into equivalent but hopefully simpler ones. In particular, the prover tries to manipulate true predicates (i.e., theorems) to *true*. In order to study a specification, we must determine which predicates to give to the theorem prover to manipulate.

For instance, if we have a rather complicated schema such as a transition table, we can ask the theorem prover to manipulate the predicate of the schema, in the hopes of getting a simpler form which we can more easily understand. This can be very useful when we are trying to figure out exactly what the system is doing, and when the predicate is written in a redundant form, such as a standard translation from a CoRE table.

We can also use the theorem prover to attempt to prove theorems such as table consistency checks. The prover may not be able to manipulate the predicate to *true*. However, if the check is indeed false, the prover may be able to strip away most of the predicate and reveal plainly the aspect of the predicate that may not be true. Then, we can inspect the simplified predicate for counterexamples; such counterexamples will also falsify the original putative theorem.

6.11 Using Z/EVES on the FGS Formal Specification

Recall that the CoRE FGS contains elements such as concurrent mode machines that depend on each other, which are prohibited in the SCR formal model. Thus many of the pleasant properties of the SCR formal model are lost. The SCR formal model requires an ordering of the variables by dependency. Thus, in this model we can perform consistency checks on tables that depend only on monitored or otherwise checked variables, so there is no chance of being forced to use an unchecked table in a consistency check of another table. This is how the SCR formal model uses consistency checks to guarantee exhaustiveness and determinism.

Consistency of the transition tables is still however a valuable and necessary property of the specification. For each transition table, we have generated the consistency check as a large number of pairwise comparison theorems, such as "if row 4's hypothesis holds, then row 7's does not". Suppose the hypothesis of row 4 is $Mode = B \land Event4$ and the hypothesis of row 7 is $Mode = C \land Event7$. Then the corresponding check will take the form

Theorem Table_Check_Four_vs_Seven:
$(Legal\_State \wedge Transition\_Tables) \Rightarrow ((Mode = B \wedge Event4) \Rightarrow \neg (Mode = C \wedge Event7))$

Since, in our FGS example, we cannot order the mode machines by dependency, we may be forced to assume tables that we do not yet know are disjoint. For instance, some rows of the *mode_Active_Vertical* transition table refer to the *mode_Vertical_Approach* transition table, but this table also refers back to the *mode_Active_Vertical* transition table. For the check to hold, certain system invariants may also have to be satisfied by the old state; these are collected in *Legal_State*.

Recall that our Z translation of a table has the property that if two rows are not disjoint, then the table is nondeterministic (as opposed to identically false, which would happen if we translate the tables as a conjunction of implications). So a consistency check using such a nondisjoint table is meaningful regardless of which of the several nondisjoint rows occurs. In contrast, if we had used possibly inconsistent table translations, then any theorem using an inconsistent table would be vacuously true, since the theorem would assume false hypotheses.

We have generated these consistency checks and we have done many experiments with proving them. Most of them are straightforward and yield to Z/EVES almost automatically. Some, however, require more effort; due to the rather large size of the Z FGS specification relative to the present capabilities of Z/EVES, some checks have not been fully investigated.

We may also wish to prove invariants, such as this invariant translated from the CoRE specification.

```
┌─ Invariant_Three ──────────────────────────────────────────
│ State
├──────────
│ (term_AP_Engaged = TRU ⇒ mode_Flight_Director ∈ FD_ON)
└────────────────────────────────────────────────────────────
```

One method of proving this theorem is to prove it inductively: show that it holds for, say, the initial state, and then that it is preserved by all transitions. In our Z FGS specification, all of the transitions are collected in *Transition_Tables*.

This theorem would take the form

Theorem Invariant_Three_Theorem:
$Invariant\_Three \wedge Transition\_Tables \Rightarrow Invariant\_Three'$

In our Z translation, we collect several invariants on *State* that we want to hold into a schema called *Legal_State*. So a check of all the invariants simultaneously would take the form

Theorem Legal_State_Theorem:
$Legal\_State \wedge Transition\_Tables \Rightarrow Legal\_State'$

For the present Z FGS specification, this theorem is not true as stated; for instance, for reasons we have discussed earlier in Section 4.5, *Invariant_Eleven* fails. We can us Z/EVES to help us construct counterexamples to candidate invariants, if they are indeed false.

We can also state a test for exhaustiveness of the entire system. This says that for any legal state and any input event, there exists a (primed) legal state such that the three are related by the predicates of *Transition_Tables*.

Theorem Exhaustiveness_Check:
$\forall Legal\_State; \ event : EVENT \bullet \exists Legal\_State' \bullet Transition\_Tables$

We can even state a test for determinism. It is as above except that it requires that the primed legal state also be unique.

Theorem Determinism_Check:
$\forall \mathit{Legal\_State};\ \mathit{event} : \mathit{EVENT} \bullet \exists_1 \mathit{Legal\_State}' \bullet \mathit{Transition\_Tables}$

We have, in some experiments, used proof attempts on these theorems to detect nondeterminism in smaller systems. However, due to the complexity of the Z FGS, which comprises several thousand lines, an analysis of this kind is at present impractical on our current hardware using the present version of Z/EVES.

We did quite a bit of testing on our Z specification. Many times, when a consistency check fails, the falsification demonstrates that there was a missing hypothesis on the state, which allowed consideration of a state that we did not want to consider legal.

6.12 Areas for Further Work

It would be very interesting to compare our results with those of the Rockwell-Collins groups working on PVS and SCR (Miller and Hoech) and SMV (Yakhnis), as well as the work at ORA of Dimitri Naydich and John Nowakowski using SPIN. Enhancements to Z/EVES and added computational power would greatly increase the feasibility of performing more substantial verification on our specification.

Chapter 7

Conclusions

7.1 The CoRE Specification

The Flight Guidance System (FGS) specified in Miller and Hoech's document [5] is intended as an example for evaluating various requirements engineering methods. Miller and Hoech used the CoRE method [1] to specify the FGS. The CoRE method can be used with the constraints of the SCR discrete-time formal model [3], but the authors chose not to use it. Unless the author one adheres to constraints such as those of the SCR formal model (e.g. a dependency ordering on the variables), he can easily write down statements whose meaning is not clear, such as the transitions for leaving `GA`, discussed in Section 4.2.

Miller and Hoech also found that the CoRE method is not entirely able to capture their intent, so their specification is expressed in a variant of CoRE that includes several of their own extensions. There are no formal semantics for these extensions, and some of them, such as the continuous transitions, are not well defined.

7.2 Using Z to Define the Semantics of the FGS Specification

In this project, we have endeavored to supply a formal semantics for the CoRE specification, using a discrete-time model based on the SCR formal model, and expressing the results in Z. We are not proposing that the FGS specification should have been written in Z in the first place; the tabular notation of CoRE is quite readable and well-suited to the FGS specification. The corresponding Z translation of the FGS specification is significantly more verbose. We have used Z as a vehicle for expressing formal definitions of concepts that appeared in the CoRE specification without formal definitions. The Z notation is quite well-suited to this task, for several reasons.

The standard Z approach to specifying transitions as constraints between an old state and a new state is very similar to the SCR discrete-time formal model's approach. We attempted to translate into Z all of the CoRE FGS notions such as events and transitions using the definitions of the SCR formal model, where an event is a change of variable between the old state and the new state of a polling cycle. If this event triggers another event, then both events occur in the same polling cycle.

The schema notation of Z allows us to assemble large systems from smaller components. We can also tailor the level of detail to the situation (such as with the *Duration(INMODE)* Booleans). Z allows us to constrain the behavior of our system as much or as little as is appropriate.

In this manner, all of the concepts admitted a reasonably straightforward translation into our Z model, except for the "continuous transition to FLC", described in Section 4.5. We wanted to preserve the property that a transition, once stated as a constraint between the old state and the new state of a polling cycle, actually held true and could not be altered in the same polling cycle. Since the intent of this "continuous transition" is to alter the new state immediately, we could not find a satisfactory translation without significantly altering the structure of the specification. The concept appears to be most suited to micro-time semantics, which we have chosen to avoid.

We found the that act of expressing the notions of the CoRE FGS specification in Z helped us to find difficulties in the original specification. For instance, while trying to formulate the concept of a simultaneous event in Z, we explored the difficulties caused by the `mode_Active_Vertical` and `mode_Active_Lateral` machines driving each other to leave the `GA` mode

7.3 Using Z/EVES on the Z Specification

Our experience has been that writing a specification, even one which is neither large nor complicated, is prone to errors (especially trivial errors of omission, misremembered and mistyped variable names, and so forth). Automated reasoning tool support can be quite efficient and cost-effective at detecting the existence of such errors, and eliminating them at an early stage of the software lifecycle.

Z/EVES provides strong, automated deduction and simplification capabilities. Type checking is done automatically. The Z/EVES user interface is well integrated with the Z notation. However, using the prover requires a good understanding of the Z notation and of predicate logic. As with most automated theorem provers, one must understand what Z/EVES does and does not do automatically, in order to provide additional guidance when Z/EVES requires it. The tutorial documentation is adequate [9].

There are some areas where Z/EVES could use significant improvement, in order to deal with state machine verification. In particular, Z/EVES support for state machine verification would be improved if Z/EVES support for free (enumerated) types were faster and more automatic.

The heuristics of Z/EVES are designed for general theorem proving and so are not especially optimized for table-checking type theorems. In particular, we had to expand schemas carefully; otherwise, we could easily introduce a large number of irrelevant hypotheses into an otherwise simple table-check, which would drastically slow down the theorem prover. As another example, there are several points at which we must disable some of the rewrite rules that Z/EVES uses, in order to keep the Z/EVES from wasting a lot of time; this is particularly true when we are arguing about membership in a subset. One must have a good idea of the type of heuristics that Z/EVES uses in order to determine what to enable or disable at a given point in a proof.

Certain kinds of errors (including otherwise minor typos) can cause spuriously true checks. For example, if the trigger for a transition is supposed to be $Mode = A \wedge Mode' \neq A$ but is accidentally entered $Mode = A \wedge Mode \neq A$, then the trigger is vacuously false. Thus all consistency checks of the form

Theorem Vacuous_Table_Check:
$(LegalState \wedge Transition\_Tables) \Rightarrow$
$((Mode = A \wedge Mode \neq A) \Rightarrow \neg\,(OtherTrigger))$

are true. An interface to a table input tool would be helpful in avoiding such errors.

7.4 Acknowledgements

First, we would like to thank Mark Saaltink, for numerous patient and thorough explanations of Z and Z/EVES. We would also like to thank Dimitri Naydich for many enlightening discussions and valuable advice. Finally, thanks to John Nowakowski for pointing out the last two examples in Section 4.6.

Bibliography

[1] Stuart R. Faulk, Lisa Finneran, James Kirby, and Assad Moini, Consortium requirements engineering guidebook, Technical Report SPC-92060-CMC, Software Productivity Consortium, 2214 Rock Hill Road, Herndon, VA 22070, December 1993.

[2] Constance Heitmeyer, Bruce Labaw, and D. Kiskis, Consistency checking of SCR–style requirements specifications, in *IEEE Internation Symposium on Requirements Engineering*, March 1995.

[3] Constance Heitmeyer, Ralph Jeffords, and Bruce Labaw, Automated Consistency Checking of Requirement Specifications, ACM Transactions on Software Engineering and Methodology, July 1996, pp. 231–261.

[4] Jonathan Jacky, *The way of Z: practical programming with formal methods*, Cambridge University Press, Cambridge, UK, 1997.

[5] Steven P. Miller and Karl F. Hoech, Specifying the Mode Logic of a Flight Guidance System in CoRE, Rockwell-Collins, April 1997.

[6] Dimitri Naydich and John Nowakowski, Flight Guidance System Validation using SPIN, ORA Technical Memo TM-97-0043, October 1997.

[7] M. Saaltink and I. Meisels, *The Z/EVES reference manual (for Version 1.4)*, ORA Canada Technical Report TR-97-5493-03c, June 1997.

[8] M. Saaltink, The Z/EVES System, in Bowen, Hinchey, and Till (eds.), "ZUM '97: The Z formal specification notation", Lec. Notes in Comp. Sci. 121, Springer-Verlag, 1997.

[9] M. Saaltink, *The Z/EVES User's Guide*, ORA Canada Technical Report TR-97-5493-06, September 1997.

[10] J. M. Spivey, *Understanding Z: a specification language and its formal semantics*, Cambridge Tracts in Theoretical Computer Science 3, Cambridge University Press, Cambridge, UK, 1988.

[11] J. M. Spivey, *The Z notation: a reference manual*, 2nd. ed., Prentice-Hall, New York, 1992.

Appendix A

Formal Specification of the FGS

This is version 1.0 (10/1/97) of the Z specification of a Flight Guidance System for Task 8 of the NASA Life-Critical Systems contract C071. This is a translation of the CoRE Flight Guidance System described in Steven P. Miller and Karl F. Hoech's document "Specifying the Mode Logic of a Flight Guidance System in CoRE," to which the reader is referred for more detailed descriptions of the variables. The order of this document follows as much as possible the order of the CoRE document, given the constraint that Z requires all variables to be declared before they are used. A few annunciations from the CoRE specification have been omitted for space reasons. Their condition tables can be checked by the methods described in the report.

A.1 Declarations of Variable Types

We enumerate the possible input events in the free type *EVENT*. Some of these comprise more than one input event; for instance, when there are two identical buttons that have the same function.

$$\begin{aligned}
EVENT ::= {} & HDG\_Switch\_Pressed \mid NAV\_Switch\_Pressed \mid APPR\_Switch\_Pressed \\
\mid {} & GA\_Pressed \mid AP\_Engage\_Switch\_Pressed \mid SYNC\_On \mid SYNC\_Off \\
\mid {} & ALT\_Switch\_Pressed \mid VS\_Switch\_Pressed \mid FLC\_Switch\_Pressed \mid FD\_Pressed \\
\mid {} & VS\_Pitch\_Wheel\_Changed \mid ALT\_Knob\_Changed \\
\mid {} & Speed\_Knob\_Changed \mid HDG\_Knob\_Changed \mid AP\_Disengage\_Pressed \\
\mid {} & AP\_Disconnect\_Bar\_Up \mid AP\_Disconnect\_Bar\_Down \\
\mid {} & Nav\_Source\_Changed \mid Land\_On\_Ground \\
\mid {} & Lateral\_NAV\_Track\_Cond\_Met \mid Lateral\_APPR\_Track\_Cond\_Met \\
\mid {} & Gone\_Overspeed \mid Gone\_Normal \mid ALTSEL\_TRACK\_Cond\_Met \\
\mid {} & ALTSEL\_CAPTURE\_Cond\_Met \\
\mid {} & Vert\_APPR\_Track\_Cond\_Met \\
\mid {} & No\_Event
\end{aligned}$$

In the following pieces of Z syntax, we define the other free types that we use in the specification. Many of these free types consist of the possible modes of various mode machines. We also define the Boolean free type, consisting of TRU and FALS, here. (This spelling accommodates Z/EVES 1.4's recognition of TRUE and FALSE as predicates).

$$\begin{array}{ll}
LATERAL\_MODE & ::= L\_GA \mid L\_APPR\_ARMED \mid L\_APPR\_TRACK \mid HDG \\
 & \mid\ ROLL\_ROLL\_HOLD \mid ROLL\_HDG\_HOLD \mid NAV\_ARMED \\
 & \mid\ NAV\_TRACK \mid LATERAL\_NOT\_IN\_MODE \\
VERTICAL\_MODE & ::= PITCH \mid V\_APPR \mid ALTSEL \mid ALTHOLD \mid V\_GA \mid VS \\
 & \mid\ FLC\_TRACK \mid FLC\_OVERSPEED \mid VERTICAL\_NOT\_IN\_MODE \\
ALTSEL\_MODE & ::= ALTSEL\_CLEARED \mid ALTSEL\_ARMED \mid ALTSEL\_CAPTURE \\
 & \mid\ ALTSEL\_TRACK \mid ALTSEL\_NOT\_IN\_MODE \\
AP\_MODE & ::= DISENGAGED\_NORMAL \mid DISENGAGED\_WARNING \\
 & \mid\ ENGAGED\_NORMAL \mid ENGAGED\_SYNC \\
VERT\_APPR\_MODE & ::= VERT\_APPR\_CLEARED \mid VERT\_APPR\_ARMED \\
 & \mid\ VERT\_APPR\_TRACK \mid VERT\_APPR\_NOT\_IN\_MODE
\end{array}$$

$$\begin{array}{ll}
LAMP\_MODE & ::= LIT \mid UNLIT \\
SPEED\_MODE & ::= SPEED\_OK \mid TOO\_FAST
\end{array}$$

$$\begin{array}{ll}
SWITCH & ::= ON \mid OFF \\
BOOLEAN & ::= TRU \mid FALS \\
VALIDITY & ::= VALID \mid INVALID \\
BAR & ::= UP \mid DOWN \\
FD\_MODE & ::= FD\_OFF \mid FD\_ON\_CUES \mid FD\_ON\_NO\_CUES \\
NAV\_TYPE & ::= FMS \mid VOR \mid LOC \\
NAV\_SOURCES & ::= FMS1 \mid FMS2 \mid FMS3 \mid VNR1 \mid VNR2 \mid VNR3 \mid VNR4 \\
AP\_COUPLING & ::= FGS \mid MANUAL
\end{array}$$

$$\begin{array}{ll}
ALTSEL\_COND & ::= ALTSEL\_COND\_CAPTURE \\
 & \mid\ ALTSEL\_COND\_TRACK \\
 & \mid\ ALTSEL\_COND\_NONE
\end{array}$$

In the following Z statement, we define some numerical types. Since there is no built-in support for real numbers, we choose precisions based on the CoRE document.

AIRSPEED is measured in 1 knot increments.
ALTITUDE is measured in 1 foot increments.
ALTITUDE_RATE is measured in 0.001 kft/min (that is, 1 ft/min) increments.
HEADING is measured in 1 degree increments.
PERIOD is measured in 1 millisecond increments.
PITCH_ANGLE is measured in 1 degree increments.
ROLL_ANGLE is measured in 1 degree increments.

$$\begin{array}{l}
AIRSPEED == \mathbb{N} \\
ALTITUDE == -8000 \mathrel{..} 56000 \\
ALTITUDE\_RATE == -32700 \mathrel{..} 32800 \\
HEADING == 0 \mathrel{..} 359 \\
MACH\_NUMBER == \mathbb{N} \\
PERIOD == \mathbb{N} \\
PITCH\_ANGLE == -90 \mathrel{..} 89 \\
ROLL\_ANGLE == -180 \mathrel{..} 179
\end{array}$$

A.2 Variable Declarations

In the schema *aggr_FCP_Switches*, we declare monitored variables for the switches on the Flight Control Panel. Each switch can assume the value *ON* or *OFF*. We also declare variables that record how many counts, or clicks, away from 0 that certain knobs are twisted. Finally, *mon_AP_Disconnect_Bar* monitors a bar that can be set to *UP* or *DOWN*, and acts as a cutoff to engaging the autopilot when *DOWN*. A change in one of these variables is associated to an input event, as described below in the event declarations.

aggr_FCP_Switches

$mon\_HDG\_Switch : SWITCH$
$mon\_NAV\_Switch : SWITCH$
$mon\_APPR\_Switch : SWITCH$
$mon\_ALT\_Switch : SWITCH$
$mon\_VS\_Switch : SWITCH$
$mon\_FLC\_Switch : SWITCH$
$mon\_AP\_Engage\_Switch : SWITCH$
$mon\_FD\_Switch\_Left : SWITCH$
$mon\_FD\_Switch\_Right : SWITCH$
$mon\_VS\_Pitch\_Count : 0 \mathrel{..} 255$
$mon\_ALT\_Count : 0 \mathrel{..} 255$
$mon\_Speed\_Count : 0 \mathrel{..} 255$
$mon\_HDG\_Count : 0 \mathrel{..} 255$
$mon\_AP\_Disconnect\_Bar : BAR$

In the following schema *aggr_FCP_Knobs*, we declare some variables that measure rotations in a given polling cycle. For instance, *term_VS_Pitch_Wheel_Rotation* measures the difference between the present value of *mon_VS_Pitch_Count* and the previous value, and so forth.

aggr_FCP_Knobs

$term\_VS\_Pitch\_Wheel\_Rotation : -128 \mathrel{..} 127$
$term\_ALT\_Knob\_Rotation : -128 \mathrel{..} 127$
$term\_Speed\_Knob\_Rotation : -128 \mathrel{..} 127$
$term\_HDG\_Knob\_Rotation : -128 \mathrel{..} 127$

In the following schema *aggr_FCP_Lamps*, we declare lamp variables that annunciate whether the FGS is in a certain mode. For instance, *con_HDG_Switch_Lamp* is *LIT* exactly when the FGS is using flight control laws that maintain a selected heading.

aggr_FCP_Lamps

$con\_HDG\_Switch\_Lamp : LAMP\_MODE$
$con\_NAV\_Switch\_Lamp : LAMP\_MODE$
$con\_APPR\_Switch\_Lamp : LAMP\_MODE$
$con\_ALT\_Switch\_Lamp : LAMP\_MODE$
$con\_VS\_Switch\_Lamp : LAMP\_MODE$
$con\_FLC\_Switch\_Lamp : LAMP\_MODE$
$con\_AP\_Engaged\_Switch\_Lamp : LAMP\_MODE$

In the following schema *SYNC*, we declare a Boolean variable *SYNC*. *term_SYNC* is *TRU* exactly when one of *mon_SYNC_Switch_Left* and *mon_SYNC_Switch_Right* is pressed. When *TRU*, *term_SYNC* causes the reference variables to be synchronized with the corresponding monitored quantities.

```
┌─ SYNC ──────────────────────────────
│ term_SYNC : BOOLEAN
└─────────────────────────────────────
```

In the following schema *aggr_Yokes_Vars*, we declare monitored variables for the controls on the pilot's and copilot's yokes. *mon_AP_Disengage_Switch_Left, mon_AP_Disengage_Switch_Right, mon_SYNC_Switch_Left, mon_SYNC_Switch_Right* are all monitored variables for switches.

```
┌─ aggr_Yokes_Vars ───────────────────
│ mon_AP_Disengage_Switch_Left : SWITCH
│ mon_AP_Disengage_Switch_Right : SWITCH
│ mon_SYNC_Switch_Left : SWITCH
│ mon_SYNC_Switch_Right : SWITCH
│ SYNC
└─────────────────────────────────────
```

In the following schema *aggr_Throttles_Vars*, we declare monitored variables for the controls on the pilot's and copilot's throttles. *mon_GA_Switch_Left* and *mon_GA_Switch_Right* are both switches.

```
┌─ aggr_Throttles_Vars ───────────────
│ mon_GA_Switch_Left : SWITCH
│ mon_GA_Switch_Right : SWITCH
└─────────────────────────────────────
```

The following schema *aggr_References* declares several term variables that are used as references to compare with monitored variables. *term_VS_Pitch_Wheel_Rotation*, *term_ALT_Knob_Rotation*, *term_Speed_Knob_Rotation*, and *term_HDG_Knob_Rotation* are each set to be half of the monitored knob count; they each range from -128 to 127 clicks.

term_Selected_Heading stores the heading, set by the crew, that the FGS uses when *mode_Active_Lateral* is in *HDG* mode or in the *ARMED* submodes of *NAV* or *L_APPR*; it is of type *HEADING* and is measured from 0 to 359 degrees in 1 degree increments.

term_Preselected_Altitude stores the altitude, set by the crew, that the FGS uses when *mode_Active_Vertical* is in *ALTSEL* mode; it is of type *ALTITUDE* and ranges from -8000 to 56000 feet in 100 foot increments.

term_Reference_IAS stores the reference indicated airspeed; it is of type *AIRSPEED*, and it ranges from 0 to 512 knots in 1 knot increments.

term_Reference_Mach stores the reference indicated Mach number. It is of type zv MACH_NUMBER and is measured from 0 to 1 Mach in 1/100 Mach increments.

term_Reference_Heading stores the reference heading. It is of type *HEADING* and is measured from 0 to 359 degrees in 1 degree increments.

term_Reference_Altitude stores the reference altitude. It is of type *ALTITUDE* and ranges from -800 to 56000 feet, in 100 foot increments.

term_Reference_Pitch stores the reference pitch. It is of type *PITCH_ANGLE* and ranges from -12 to 20 degrees in 1 degree increments.

term_Reference_Roll stores the reference roll. It is of type *ROLL_ANGLE* and ranges from -180 to 180 degrees in 1 degree increments.

term_Reference_Vertical_Speed stores the reference vertical speed. It is of type *ALTITUDE_RATE* and ranges from -5 to 5 kft/min in 0.1 kft/min increments; thus it is measured in multiples of 100 of the 0.001 kft/min increments of *ALTITUDE_RATE*.

$$
\begin{array}{|l}
\textit{aggr\_References} \\
\hline
\textit{term\_Preselected\_Altitude} : \textit{ALTITUDE} \\
\textit{term\_Selected\_Heading} : \textit{HEADING} \\
\textit{term\_Reference\_IAS} : \textit{AIRSPEED} \\
\textit{term\_Reference\_Mach} : \textit{MACH\_NUMBER} \\
\textit{term\_Reference\_Heading} : \textit{HEADING} \\
\textit{term\_Reference\_Altitude} : \textit{ALTITUDE} \\
\textit{term\_Reference\_Pitch} : \textit{PITCH\_ANGLE} \\
\textit{term\_Reference\_Roll} : \textit{ROLL\_ANGLE} \\
\textit{term\_Reference\_Vertical\_Speed} : \textit{ALTITUDE\_RATE} \\
\hline
\textit{term\_Preselected\_Altitude} \in \{ i : \textit{ALTITUDE} \bullet 100 * i \} \\
\textit{term\_Reference\_IAS} \in 0 \,..\, 512 \\
\textit{term\_Reference\_Mach} \in 0 \,..\, 100 \\
\textit{term\_Reference\_Altitude} \in \{ i : \textit{ALTITUDE} \bullet 100 * i \} \\
\textit{term\_Reference\_Pitch} \in -12 \,..\, 20 \\
\textit{term\_Reference\_Roll} \in -27 \,..\, 27 \\
\textit{term\_Reference\_Vertical\_Speed} \in \{ i : \textit{ALTITUDE\_RATE} \bullet 100 * i \} \cap (-5000 \,..\, 5000) \\
\hline
\end{array}
$$

The schema *aggr_Reference_Annunciations* declares variables for the console annunciations to the crew.

$$
\begin{array}{|l}
\textit{aggr\_Reference\_Annunciations} \\
\hline
\textit{con\_Selected\_Heading\_Annunciation} : \textit{HEADING} \\
\textit{con\_Preselected\_Altitude\_Annunciation} : \textit{ALTITUDE} \\
\textit{con\_Reference\_IAS\_Annunciation} : \textit{AIRSPEED} \\
\textit{con\_Reference\_VS\_Annunciation} : \textit{ALTITUDE\_RATE} \\
\hline
\textit{con\_Reference\_IAS\_Annunciation} \in 0 \,..\, 512 \\
\textit{con\_Reference\_VS\_Annunciation} \in \{ i : \textit{ALTITUDE\_RATE} \bullet 100 * i \} \cap (-5000 \,..\, 5000) \\
\hline
\end{array}
$$

The following schema *INMODE_Booleans* declares some Booleans that indicate whether certain mode machines have been in a certain mode for longer than a preset amount of time. (*gt* stands for "greater than".)

```
INMODE_Booleans
Duration_INMODE_AP_Disengaged_Warning_gt_ten_sec : BOOLEAN
Duration_INMODE_Vert_Appr_Track_gt_const_min_armed_period : BOOLEAN
Duration_INMODE_NAV_ARMED_gt_const_min_armed_period : BOOLEAN
Duration_INMODE_APPR_ARMED_gt_const_min_armed_period : BOOLEAN
Duration_INMODE_ALTSEL_ARMED_gt_const_min_armed_period : BOOLEAN
Duration_INMODE_ALTSEL_CAPT_gt_const_min_armed_period : BOOLEAN
```

The following schema *aggr_Air_Data* declares several monitored variables.

mon_Indicated_Airspeed, the indicated airspeed as measured by comparing the ram (pitot) air pressure with the static air pressure, can range from 0 to 512 knots.

mon_Indicated_Mach_Number, the ratio of true airspeed to the speed of sound, can range from 0 to 1 Mach in 1/100 Mach increments. *mon_Indicated_Altitude*, computed from static air pressure and corrected for local ambient pressure conditions can range from -8000 to 56000 feet, in 1 foot increments.

mon_Pressure_Altitude, the altitude computed from the static air pressure assuming standard pressure conditions, is of type *ALTITUDE* and can range from -8000 to 56000 feet, in 1 foot increments.

mon_Roll_Angle, the roll angle measured in degrees from wings level, with positive value indicating that the right wing is down, is of type *ROLL_ANGLE* and can range from -180 to 180 degrees in 1 degree increments.

mon_Pitch_Angle, the pitch angle measured in degrees nose-up from level, is of type *PITCH_-ANGLE* and can range from -90 to 90 degrees in 1 degree increments.

mon_Vertical_Speed, the vertical speed of the aircraft computed by comparing instantaneous air pressure with that filtered through a diaphragm with a calibrated leak, is of type *ALTITUDE_RATE* and can range from -32.7 to 32.7 kft/min, in 0.016 kft/min increments; it is thus measured in multiples of 16 of the 0.001 kft/min increments of the type *ALTITUDE_RATE*.

mon_Heading is the aircraft's heading, measured in degrees clockwise from magnetic North. It is of type *HEADING* and ranges from 0 to 359 degrees in 1 degree increments.

mon_On_Ground is va Boolean that is *TRU* exactly when the aircraft is on the ground.

term_Above_Transition_Altitude is a Boolean that is *TRU* exactly when the the aircraft's altitude is above a certain constant.

term_Overspeed is Boolean that is *TRU* exactly when a certain mode machine, *mode_Overspeed* has the mode *TOO_FAST*, which occurs when the aircraft exceeds its maximum operating speed.

$aggr\_Air\_Data$

$mon\_Indicated\_Airspeed : AIRSPEED$
$mon\_Indicated\_Mach\_Number : MACH\_NUMBER$
$mon\_Indicated\_Altitude : ALTITUDE$
$mon\_On\_Ground : BOOLEAN$
$term\_Above\_Transition\_Altitude : BOOLEAN$
$term\_Overspeed : BOOLEAN$
$mon\_Pressure\_Altitude : ALTITUDE$
$mon\_Roll\_Angle : ROLL\_ANGLE$
$mon\_Pitch\_Angle : PITCH\_ANGLE$
$mon\_Vertical\_Speed : ALTITUDE\_RATE$
$mon\_Heading : HEADING$

$mon\_Indicated\_Airspeed \in 0 \mathrel{..} 512$
$mon\_Indicated\_Mach\_Number \in 0 \mathrel{..} 100$
$mon\_Vertical\_Speed \in \{i : ALTITUDE\_RATE \bullet 16 * i\}$

The following axiomatic definition declares the maximum safe speed of the aircraft, measured in knots (*term_Vmo*) and in Mach number (*term_Mmo*). We also declare the constant *const_Transition_Altitude*, above which the aircraft uses Mach number to measure airspeed and pressure altitude, and below which the aircraft uses indicated airspeed and indicated altitude. This constant equals 18000 ft.

$term\_Vmo : \mathbb{N}$
$term\_Mmo : \mathbb{N}$
$const\_Transition\_Altitude : \mathbb{N}$

$const\_Transition\_Altitude = 18000$

The following schema *Overspeed* declares the *mode_Overspeed* mode machine. This mode machine has two modes: *SPEED_OK* when the aircraft is flying at a safe operating speed, and *TOO_FAST* when the aircraft is flying at too high a speed.

$Overspeed$

$mode\_Overspeed : SPEED\_MODE$

$Init\_Of\_Overspeed$

$Overspeed$

$mode\_Overspeed = SPEED\_OK$

The following schema *aggr_Nav_Source_Data* declares some navigation variables. *term_Selected_Nav_Type* stores what kind of navigational device is currently selected: *FMS* if the selected source is a Flight Management System, *VOR* if the selected source is a VHF omnirange navigation beacon, and *LOC* if the selected navigation source is an ILS localizer beacon. *mon_Selected_Nav_Source_Status* is *VALID* precisely when the navigational source is generating valid data, and *INVALID* otherwise. Finally, the monitored Boolean *mon_Valid_Glideslope* is *TRU* precisely when the FGS is receiving a valid ILS glideslope signal, and *FALS* otherwise.

```
┌─ aggr_Nav_Source_Data ─────────────────────────
│ term_Selected_Nav_Type : NAV_TYPE
│ mon_Selected_Nav_Source_Status : VALIDITY
│ mon_Valid_Glideslope : BOOLEAN
└────────────────────────────────────────────────
```

The following schema *aggr_Nav_Source_Mons* declares the navigational monitored variables. The *mon_Selected_Nav_Source* can be one of seven values: *FMS1, FMS2, FMS3* indicate that the first, second, or third Flight Management system is the selected navigation source, and *VNR1, VNR2, VNR3, VNR4* indicate that the first, second, third, or fourth VHF Navigation Receiver is the selected navigation source.

mon_Selected_Nav_Source_Frequency is a partial function from *NAV_SOURCES* to the range [108.0, 136.0] MHz (measured in 0.1 megahertz increments). This function sends a source in *VNR1*, ..., *VNR4* to the carrier frequency to which it is tuned.

mon_Nav_Source_Signal_Type is a partial function from *NAV_SOURCES* to *NAV_TYPE*. This function sends a source in *VNR1*, ..., *VNR4* to the type of frequency (*VOR, LOC*) to which the VHF Navigation Receiver is tuned.

```
┌─ aggr_Nav_Source_Mons ─────────────────────────
│ mon_Selected_Nav_Source : NAV_SOURCES
│ mon_Selected_Nav_Source_Frequency : NAV_SOURCES ⇸ 1080 .. 1360
│ mon_Nav_Source_Signal_Type : NAV_SOURCES ⇸ NAV_TYPE
└────────────────────────────────────────────────
```

The following schema *Autopilot* declares the *mode_Autopilot* mode machine. This machine can be in the following two mode (each of which has submodes): *ENGAGED*, when the autopilot is engaged and processing the flight control laws, *DISENGAGED*, when the autopilot is not engaged.

The *AP_Engaged_Mode* machine has two values when *mode_Autopilot* = *ENGAGED*: *AP_ENGAGED_NORMAL* when the SYNC switch is not pressed; and *AP_ENGAGED_SYNC* when the SYNC switch is pressed (and thus the autopilot is temporarily disengaged); when *mode_Autopilot* $\neq$ *ENGAGED* then *AP_Engaged_Mode* = *AP_ENGAGED_NOT_IN_MODE*.

The *AP_Disengaged_Mode* machine has two values when *mode_Autopilot* = *DISENGAGED*: *AP_DISENGAGED_WARNING*, when *mode_Autopilot* has recently switched from *ENGAGED* to *DISENGAGED* and the autopilot is still functioning; and *AP_DISENGAGED_NORMAL*, once *mode_Autopilot* = *DISENGAGED* more than 10 seconds, and the autopilot actually disenages. If *mode_Autopilot* $\neq$ *DISENGAGED* then *AP_Disengaged_Mode* = *AP_DISENGAGED_NOT_-IN_MODE*.

The *mon_AP_Disconnect_Bar* can be *UP* (when the crew has not pressed down the autopilot disconnect bar) or *DOWN* (when the crew has pressed the bar, which forces the autopilot to to be disengaged).

The Boolean *term_AP_Engaged* is *TRU* exactly when *mode_Autopilot* is *ENGAGED*, and *FALS* otherwise.

The monitored variable *con_AP_Coupling* has two values: when set to *FGS*, the command input to the control surfaces of the aircraft is generated by the FGS; when set to *MANUAL*, the command input to the control surfaces of the aircraft is manually generated by the pilots using the control yoke and rudder pedals.

The monitored switch *con_AP_Disengage_Warning* has two values: when set to *ON*, the disengagement of the autopilot is signaled to the flight crew by sounding the EICAS autopilot disengagemant aural warning and by the presence of a visual warning on the EFIS. When set to *OFF*, then disengagement of the autpilot is not being signaled to the flight crew.

The *AP_Disengaged_Warning_Clock* is set to 0 whenever *AP_Disengaged_Mode* enters *AP_DISENGAGED_WARNING* mode, and is incremented each cycle that *AP_Disengaged_Mode* = *AP_DISENGAGED_WARNING*.

Autopilot

mode_Autopilot : *AP_MODE*
mon_AP_Disconnect_Bar : *BAR*
term_AP_Engaged : *BOOLEAN*
con_AP_Coupling : *AP_COUPLING*
con_AP_Disengage_Warning : *SWITCH*

Init_Of_Autopilot

Autopilot

mode_Autopilot = *DISENGAGED_NORMAL*

The following schema *Lateral_Terms* defines some terms associated with the lateral mode. The Boolean *term_Within_Lateral_NAV_Capture_Window* is *TRU* exactly when the aircraft has the appropriate operating conditions (e.g. altitude, speed, position, and heading) to make a safe capture of the navigation track, and *FALS* otherwise. The precise definition depends on the customer.

The Boolean *term_Lateral_NAV_Track_Cond_Met* is *TRU* exactly when the aircraft and its system satisfy all conditions necessary for tracking a lateral navigation source, and *FALS* otherwise.

The Boolean *term_Within_Lateral_APPR_Capture_Window* is *TRU* exactly when the aircraft has the appropriate operating conditions (e.g. altitude, speed, position, and heading) to make a safe capture of the approach track, and *FALS* otherwise. The precise definition depends on the customer.

The Boolean *term_Lateral_NAV_Track_Cond_Met* is *TRU* exactly when the aircraft and its system satisfy all conditions necessary for tracking a precision lateral approach source, and *FALS* otherwise.

The Boolean *term_Roll_LE_Threshold* is *TRU* exactly when the aircraft's roll angle is less than or equal to the constant *const_Rool_Selection_Threshold*.

The constant *const_Roll_Selection_Threshold* marks the boundary between the selection of the *ROLL_HOLD* and *HDG_HOLD* modes of the submode *Roll_Mode*.

Lateral_Terms

term_Within_Lateral_NAV_Capture_Window : *BOOLEAN*
term_Lateral_NAV_Track_Cond_Met : *BOOLEAN*
term_Within_Lateral_APPR_Capture_Window : *BOOLEAN*
term_Lateral_APPR_Track_Cond_Met : *BOOLEAN*
term_Roll_LE_Threshold : *BOOLEAN*
const_Roll_Selection_Threshold : $\mathbb{N}$

const_Roll_Selection_Threshold = 50

The following schema *Vertical_Terms* declares some term variables relevant to the *mode_Active_Vertical* mode machine. The Boolean *term_Within_Vertical_APPR_Capture_Window* is *TRU* exactly when the aircraft has the appropriate operating conditions (e.g. altitude, speed, position, and heading) to make a safe capture of the approach track, and *FALS* otherwise. The precise definition depends on the customer.

The Boolean *term_VerticalAPPR_Track_Cond_Met* is *TRU* exactly when the aircraft and its system satisfy all conditions necessary for tracking a precision vertical approach source, and *FALS* otherwise.

$$
\begin{array}{|l}
\textit{Vertical\_Terms} \\
\hline
term\_Within\_Vertical\_APPR\_Capture\_Window : BOOLEAN \\
term\_Vertical\_APPR\_Track\_Cond\_Met : BOOLEAN \\
\hline
\end{array}
$$

The following schema *AltselTerms* declares the term variable *term_ALTSEL_Cond.* This variable has three possible values: *ALTSEL_CAPTURE, ALTSEL_TRACK, ALTSEL_NONE.* It takes the values *ALTSEL_CAPTURE* and *ALTSEL_TRACK* exactly when the aircraft has the appropriate operating conditions (e.g. altitude, speed, position, and heading) relative to *term_Preselected_Altitude* to make a safe capture and safe track, respectively. The precise definition depends on the custome

$$
\begin{array}{|l}
\textit{Altsel\_Terms} \\
\hline
term\_ALTSEL\_Cond : ALTSEL\_COND \\
\hline
\end{array}
$$

The following axiomatic definition declares display constants.

const_annunciation_update_deadline stores the maximum amount of time allowed for a reference annunciation (such as *term_Reference_Pitch*) to respond to an event; it is of type *PERIOD* and is measured in milliseconds.

const_blink_time is the amount of time a mode annunciation blinks upon change; it is of type *PERIOD* and is measured in milliseconds.

const_display_update_deadline stores the maximum amount of time allowed for a flight display or indicator lamp to respond to an event; it is of type *PERIOD* and is measured in milliseconds.

$$
\begin{array}{|l}
const\_annunc\_update\_deadline : PERIOD \\
const\_blink\_time : PERIOD \\
const\_display\_update\_deadline : PERIOD \\
\hline
const\_annunc\_update\_deadline = 100 \\
const\_blink\_time = 5000 \\
const\_display\_update\_deadline = 100
\end{array}
$$

A.3 Collections of Events

In the following axiomatic definitions, we enumerate some useful subsets of events. For the modes *mode_Active_Vertical* and *mode_Active_Lateral*, we list all of the input events that trigger transitions in their tables, for use in separating some transitions as discussed in section 4.3. These set definitions are labeled as rewrite rules for Z/EVES so that they can be used in proofs if necessary.

$Vertical\_Events : \mathbb{P}\, EVENT$

[rule Vertical_Events_Rule]
$Vertical\_Events = \{SYNC\_On, VS\_Pitch\_Wheel\_Changed,$
$ALTSEL\_CAPTURE\_Cond\_Met, ALT\_Knob\_Changed, AP\_Engage\_Switch\_Pressed,$
$ALT\_Switch\_Pressed, VS\_Switch\_Pressed,$
$FLC\_Switch\_Pressed, Vert\_APPR\_Track\_Cond\_Met,$
$GA\_Pressed, SYNC\_On\}$

$Lateral\_Events : \mathbb{P}\, EVENT$

[rule Lateral_Events_Rule]
$Lateral\_Events = \{HDG\_Switch\_Pressed, NAV\_Switch\_Pressed,$
$Nav\_Source\_Changed, APPR\_Switch\_Pressed, AP\_Engage\_Switch\_Pressed, SYNC\_On\}$

The subsets *Flight_Mode_Requested*, *Lateral_Mode_Requested*, and *Vertical_Mode_Requested* are useful in the specification of the mode machine transitions, as discussed in section 6.4. These set definitions are labeled as rewrite rules for Z/EVES so that they can be used in proofs if necessary.

$Flight\_Mode\_Requested : \mathbb{P}\, EVENT$

[rule Flight_Mode_Requested_Rule]
$Flight\_Mode\_Requested = \{HDG\_Switch\_Pressed, NAV\_Switch\_Pressed,$
$APPR\_Switch\_Pressed, VS\_Switch\_Pressed, ALT\_Switch\_Pressed,$
$FLC\_Switch\_Pressed, GA\_Pressed\}$

$Lateral\_Mode\_Requested : \mathbb{P}\, EVENT$

[rule Lateral_Mode_Requested_Rule]
$Lateral\_Mode\_Requested = \{HDG\_Switch\_Pressed, NAV\_Switch\_Pressed,$
$APPR\_Switch\_Pressed, GA\_Pressed\}$

$Vertical\_Mode\_Requested : \mathbb{P}\, EVENT$

[rule Vertical_Mode_Requested_Rule]
$Vertical\_Mode\_Requested = \{VS\_Switch\_Pressed,$
$ALT\_Switch\_Pressed, FLC\_Switch\_Pressed, GA\_Pressed\}$

A.4 Collections of modes

In the following axiomatic definitions, we define some collections of modes that are used to simulate the hierarchical mode machines of the CoRE specification, as discussed in 6.6. A CoRE expression such as `mode_Flight_Director = ON` is replaced by a Z expression $mode\_Flight\_Director \in FD\_ON$.

$FD\_ON : \mathbb{P}\, FD\_MODE$

[rule FD_ON_Rule]
$FD\_ON = \{FD\_ON\_CUES, FD\_ON\_NO\_CUES\}$

$ENGAGED, DISENGAGED : \mathbb{P}\, AP\_MODE$

[rule ENGAGED_Rule]
$ENGAGED = \{ENGAGED\_NORMAL, ENGAGED\_SYNC\}$

[rule DISENGAGED_Rule]
$DISENGAGED = \{DISENGAGED\_NORMAL, DISENGAGED\_WARNING\}$

$ROLL : \mathbb{P}\, LATERAL\_MODE$

[rule ROLL_MODE_Rule]
$ROLL = \{ROLL\_ROLL\_HOLD, ROLL\_HDG\_HOLD\}$

$NAV : \mathbb{P}\, LATERAL\_MODE$

[rule NAV_MODE_Rule]
$NAV = \{NAV\_ARMED, NAV\_TRACK\}$

$L\_APPR : \mathbb{P}\, LATERAL\_MODE$

[rule L_APPR_MODE_Rule]
$L\_APPR = \{L\_APPR\_ARMED, L\_APPR\_TRACK\}$

$FLC : \mathbb{P}\, VERTICAL\_MODE$

[rule FLC_Rule]
$FLC = \{FLC\_TRACK, FLC\_OVERSPEED\}$

$ALTSEL\_ENABLED, ALTSEL\_ACTIVE : \mathbb{P}\, ALTSEL\_MODE$

[rule ALTSEL_ENABLED_Rule]
$ALTSEL\_ENABLED = \{ALTSEL\_ARMED, ALTSEL\_CAPTURE, ALTSEL\_TRACK\}$

[rule ALTSEL_ACTIVE_Rule]
$ALTSEL\_ACTIVE = \{ALTSEL\_CAPTURE, ALTSEL\_TRACK\}$

$VERT\_APPR\_ENABLED : \mathbb{P}\, VERT\_APPR\_MODE$

[rule VERT_APPR_ENABLED_Rule]
$VERT\_APPR\_ENABLED = \{VERT\_APPR\_ARMED, VERT\_APPR\_TRACK\}$

A.5 Flight Mode Declarations

The following schema *Flight_Director* declares a mode machine *mode_Flight_Director*. It has three modes: *FD_OFF, FD_ON_CUES, FD_ON_NO_CUES*. It is in mode *FD_OFF* exactly when the Flight Director is off. It is in mode *FD_ON_CUES* exactly when the Flight Director is on and annunciating cues to the crew, and it is in mode *FD_ON_NO_CUES* exactly when the Flight Director is on but not annunciating to the crew.

Flight_Director

mode_Flight_Director : *FD_MODE*

The following four schemas *Active_Lateral*, *Active_Vertical*, *Altitude_Select*, and *Vertical_Approach* declare the principal mode machines of the FGS. The *NOT_IN_MODE* modes are used to simulate the CoRE sustaining condition imposed on them, which is that `mode_Flight_Director = ON`.

The following schema *Active_Lateral* declares a mode machine *mode_Active_Lateral*. This mode machine has the following modes: *HDG, ROLL_ROLL_HOLD, ROLL_HDG_HOLD, NAV_ARMED, NAV_TRACK, L_GA, L_APPR_ARMED, L_APPR_TRACK, LATERAL_NOT_IN_MODE*.

When *mode_Active_Lateral* is in *HDG*, the FGS generates commands to capture and maintain the selected heading.

When *mode_Active_Lateral* is in *ROLL_ROLL_HOLD*, the FGS generates commands to maintain the reference roll of the aircraft.

When *mode_Active_Lateral* is in *ROLL_HDG_HOLD*, then FGS generates commands to maintain the reference heading of the aircraft.

When *mode_Active_Lateral* is in *NAV_ARMED*, the FGS generates commands to capture and maintain the selected heading until an external navigation source such as a VOR, LOC, or FMS can be captured.

When *mode_Active_Lateral* is in *NAV_TRACK*, the FGS generates commands to capture and track and external navigation source such as a VOR, LOC, or FMS.

When *mode_Active_Lateral* is in *L_GA*, the FGS generates commands to perform a Go Around operation.

When *mode_Active_Lateral* is in *L_APPR_ARMED*, the aircraft generates commands to capture and maintain the selected heading until a precision navigation source such as LOC can be detected.

When *mode_Active_Lateral* is in *L_APPR_TRACK*, the FGS generates commands to capture and track a precision navigation source such as a LOC.

When *mode_Active_Lateral* is in *NOT_IN_MODE*, the Flight Director is off.

Active_Lateral

mode_Active_Lateral : *LATERAL_MODE*

The following schema *Active_Vertical* declares a mode machine *mode_Active_Vertical*. This mode machine has the following modes: *PITCH, V_APPR, ALTSEL, ALTHOLD, V_GA, VS, FLC_TRACK, FLC_OVERSPEED, VERTICAL_NOT_IN_MODE*.

When *mode_Active_Vertical* is in *PITCH*, the FGS generates commands to maintain the reference pitch.

When *mode_Active_Vertical* is in *V_APPR*, the behavior of the FGS is determined by the *mode_Vertical_Approach* mode machine.

When *mode_Active_Vertical* is in *ALTSEL*, the behavior of the FGS is determined by the *mode_Altitude_Select* mode machine.

When *mode_Active_Vertical* is in *ALTHOLD*, the FGS generates pitch commands to maintain the reference altitude.

When *mode_Active_Vertical* is in *VS*, the FGS generates pitch commands to maintain the reference vertical speed.

When *mode_Active_Vertical* is in *FLC_TRACK*, the FGS generates commands to acquire and track the reference airspeed (or reference mach number, depending on the altitude), taking into account the pilot's intent to climb or descend as indicated by the preselected altitude and the aircraft's ability to accomplish that intent.

When *mode_Active_Vertical* is in *FLC_OVERSPEED* the FGS generates pitch commands to acquire an airspeed or Mach number slightly below the maximum operating airspeed (V_{mo} or M_{mo}).

When *mode_Active_Vertical* is in *VERTICAL_NOT_IN_MODE*, the Flight Director is off.

```
── Active_Vertical ──────────────────────────────
│ mode_Active_Vertical : VERTICAL_MODE
└─────────────────────────────────────────────────
```

The following schema *Altitude_Select* declares a mode machine *mode_Altitude_Select.* This mode machine has the following modes: *ALTSEL_CLEARED, ALTSEL_ARMED, ALTSEL_CAPTURE, ALTSEL_TRACK, ALTSEL_NOT_IN_MODE.*

When *mode_Altitude_Select* is in *ALTSEL_CLEARED*, the FGS generates no commands to monitor, capture, or track the preselected altitude; vertical guidance commands are generated using the *mode_Active_Vertical* mode machine.

When *mode_Altitude_Select* is in *ALTSEL_ARMED*, the FGS monitors the aircraft closure rate towards the preselected altitude and determines the optimum capture point; vertical guidance commands are generated using the *mode_Active_Vertical* mode machine.

When *mode_Altitude_Select* is in *ALTSEL_CAPTURE*, the FGS generated commands for a smooth, low-acceleration acquisition of the preselected altitude.

When *mode_Altitude_Select* is in *ALTSEL_TRACK*, the FGS generates commands to maintain the preslected altitude.

When *mode_Altitude_Select* is in *ALTSEL_NOT_IN_MODE*, the Flight Director is off.

```
── Altitude_Select ──────────────────────────────
│ mode_Altitude_Select : ALTSEL_MODE
└─────────────────────────────────────────────────
```

The following schema *Vertical_Approach* declares a mode machine *mode_Vertical_Approach.* This mode machine has the following modes: *VERT_APPR_CLEARED, VERT_APPR_ARMED, VERT_APPR_TRACK, VERT_APPR_NOT_IN_MODE.*

When in *VERT_APPR_CLEARED*, the FGS generates no commands to track or monitor a vertical guidance source; vertical guidance commands are generated using the *mode_Active_Vertical* mode machine.

When in *VERT_APPR_ARMED*, the FGS monitors aircraft closure towards the approach glideslope (if ILS) or glide-path (if FMS) and determines the optimum capture point; vertical guidance commands are generated using the *mode_Active_Vertical* mode machine.

When in *VERT_APPR_TRACK*, the FGS generates pitch commands to capture and track the glideslope (if ILS) or glide-path (if FMS).

When in *VERT_APPR_NOT_IN_MODE*, the Flight Director is off.

```
── Vertical_Approach ────────────────────────────
│ mode_Vertical_Approach : VERT_APPR_MODE
└─────────────────────────────────────────────────
```

The schema *aggr_Flight_Modes* collects the flight modes:

```
── aggr_Flight_Modes ──────────────────────────────
│ Flight_Director
│ Active_Lateral
│ Active_Vertical
│ Altitude_Select
│ Vertical_Approach
└──────────────────────────────────────────────────
```

A.6 The FGS State

The schema *Variables* collects the variables together in a large schema that we will use in our declaration of *State*:

```
── Variables ──────────────────────────────────────
│ aggr_FCP_Lamps
│ aggr_FCP_Switches
│ aggr_FCP_Knobs
│ aggr_Yokes_Vars
│ aggr_Throttles_Vars
│ aggr_References
│ aggr_Reference_Annunciations
│ aggr_Air_Data
│ Overspeed
│ aggr_Nav_Source_Data
│ aggr_Nav_Source_Mons
│ Autopilot
│ Lateral_Terms
│ Vertical_Terms
│ Altsel_Terms
│ INMODE_Booleans
│ aggr_Flight_Modes
└──────────────────────────────────────────────────
```

The schema *The_Event* defines the variable *event*, which carries the input event of a polling cycle:

```
── The_Event ──────────────────────────────────────
│ event : EVENT
└──────────────────────────────────────────────────
```

A.7 Definitions of Some Input Events and Terms

In the following schemas, we define the input events. Each input event corresponds to some monitored variable (or combination of monitored variables) changing its value during a polling cycle. All changes of monitored variables that trigger transitions are listed here. Since the variable *event* can only have one value during a given polling cycle, the One Input Assumption is effectively guaranteed by this construction. Some input event definitions are given later, because of the order in which they appear in the CoRE specification.

Event_HDG_Switch_Pressed

$\Delta aggr\_FCP\_Switches$
The_Event

$event = HDG\_Switch\_Pressed \Leftrightarrow$
$mon\_HDG\_Switch = OFF \land mon\_HDG\_Switch' = ON$

Event_NAV_Switch_Pressed

$\Delta aggr\_FCP\_Switches$
The_Event

$event = NAV\_Switch\_Pressed \Leftrightarrow$
$mon\_NAV\_Switch = OFF \land mon\_NAV\_Switch' = ON$

Event_APPR_Switch_Pressed

$\Delta aggr\_FCP\_Switches$
The_Event

$event = APPR\_Switch\_Pressed \Leftrightarrow$
$(mon\_APPR\_Switch = OFF \land mon\_APPR\_Switch' = ON)$

Event_ALT_Switch_Pressed

$\Delta aggr\_FCP\_Switches$
The_Event

$event = ALT\_Switch\_Pressed \Leftrightarrow$
$(mon\_ALT\_Switch = OFF \land mon\_ALT\_Switch' = ON)$

Event_FLC_Switch_Pressed

$\Delta aggr\_FCP\_Switches$
The_Event

$event = FLC\_Switch\_Pressed \Leftrightarrow$
$(mon\_FLC\_Switch = OFF \land mon\_FLC\_Switch' = ON)$

Event_VS_Switch_Pressed

$\Delta aggr\_FCP\_Switches$
The_Event

$event = VS\_Switch\_Pressed \Leftrightarrow$
$(mon\_VS\_Switch = OFF \land mon\_VS\_Switch' = ON)$

Def_Of_term_VS_Pitch_Wheel_Rotation

$\Delta aggr\_FCP\_Knobs$
$\Delta aggr\_FCP\_Switches$

$term\_VS\_Pitch\_Wheel\_Rotation' =$
$((mon\_VS\_Pitch\_Count' - mon\_VS\_Pitch\_Count + 128) \bmod 256) - 128$

$Event\_VS\_Pitch\_Wheel\_Changed$

The_Event
$\Delta aggr\_FCP\_Switches$

$event = VS\_Pitch\_Wheel\_Changed \Leftrightarrow$
$(mon\_VS\_Pitch\_Count \neq mon\_VS\_Pitch\_Count')$

$Def\_Of\_term\_ALT\_Knob\_Rotation$

$\Delta aggr\_FCP\_Knobs$
$\Delta aggr\_FCP\_Switches$

$term\_ALT\_Knob\_Rotation' =$
$((mon\_ALT\_Count' - mon\_ALT\_Count + 128) \bmod 256) - 128$

$Event\_ALT\_Knob\_Changed$

The_Event
$\Delta aggr\_FCP\_Switches$

$event = ALT\_Knob\_Changed \Leftrightarrow$
$(mon\_ALT\_Count \neq mon\_ALT\_Count')$

$Def\_Of\_term\_Speed\_Knob\_Rotation$

$\Delta aggr\_FCP\_Knobs$
$\Delta aggr\_FCP\_Switches$

$term\_Speed\_Knob\_Rotation' =$
$((mon\_Speed\_Count' - mon\_Speed\_Count + 128) \bmod 256) - 128$

$Event\_Speed\_Knob\_Changed$

The_Event
$\Delta aggr\_FCP\_Switches$

$event = Speed\_Knob\_Changed \Leftrightarrow$
$(mon\_Speed\_Count \neq mon\_Speed\_Count')$

$Def\_Of\_term\_HDG\_Knob\_Rotation$

$\Delta aggr\_FCP\_Knobs$
$\Delta aggr\_FCP\_Switches$

$term\_HDG\_Knob\_Rotation' =$
$((mon\_HDG\_Count' - mon\_HDG\_Count + 128) \bmod 256) - 128$

$Event\_HDG\_Knob\_Changed$

The_Event
$\Delta aggr\_FCP\_Switches$

$event = HDG\_Knob\_Changed \Leftrightarrow$
$(mon\_HDG\_Count \neq mon\_HDG\_Count')$

$Event\_AP\_Engage\_Switch\_Pressed$

$\Delta aggr\_FCP\_Switches$
$\Delta Autopilot$
The_Event

$event = AP\_Engage\_Switch\_Pressed \Leftrightarrow$
$(mon\_AP\_Engage\_Switch = OFF \land mon\_AP\_Engage\_Switch' = ON)$

$Event\_FD\_Pressed$

The_Event
$\Delta aggr\_FCP\_Switches$

$event = FD\_Pressed \Leftrightarrow$
$((mon\_FD\_Switch\_Left = OFF \land mon\_FD\_Switch\_Left' = ON)$
$\lor (mon\_FD\_Switch\_Right = OFF \land mon\_FD\_Switch\_Right' = ON))$

$Event\_AP\_Disengage\_Pressed$

$\Delta Autopilot$
$\Delta aggr\_Yokes\_Vars$
The_Event

$event = AP\_Disengage\_Pressed \Leftrightarrow$
$((mon\_AP\_Disengage\_Switch\_Left = OFF$
$\land mon\_AP\_Disengage\_Switch\_Left' = ON) \lor$
$(mon\_AP\_Disengage\_Switch\_Right = OFF$
$\land mon\_AP\_Disengage\_Switch\_Right' = ON))$

$Def\_Of\_term\_SYNC$

$SYNC$
$aggr\_Yokes\_Vars$

$term\_SYNC = TRU \Leftrightarrow$
$(mon\_SYNC\_Switch\_Left = ON \lor mon\_SYNC\_Switch\_Right = ON)$

$Event\_GA\_Pressed$

$\Delta aggr\_Throttles\_Vars$
The_Event

$event = GA\_Pressed \Leftrightarrow$
$((mon\_GA\_Switch\_Left = OFF \land mon\_GA\_Switch\_Left' = ON) \lor$
$(mon\_GA\_Switch\_Right = OFF \land mon\_GA\_Switch\_Right' = ON))$

A.8 Event Tables

$Def\_Of\_term\_Selected\_Heading$

$\Delta aggr\_References$
$\Delta aggr\_FCP\_Knobs$
The_Event

$event = HDG\_Knob\_Changed \wedge$
$(term\_Selected\_Heading' =$
$(term\_Selected\_Heading + term\_HDG\_Knob\_Rotation' \bmod 360))$
$\vee ((\neg\ event = HDG\_Knob\_Changed) \wedge term\_Selected\_Heading' =$
$term\_Selected\_Heading)$

$Init\_Of\_term\_Selected\_Heading$

$aggr\_Air\_Data$
$aggr\_References$

$term\_Selected\_Heading = mon\_Heading$

$Def\_Of\_term\_Preselected\_Altitude$

$\Delta aggr\_References$
The_Event
$\Delta aggr\_FCP\_Knobs$

$(event = ALT\_Knob\_Changed \wedge term\_Preselected\_Altitude' =$
$min(\{36000, max(\{0, term\_Preselected\_Altitude + 100 * term\_ALT\_Knob\_Rotation'\})\})) \vee$
$(\neg\ (event = ALT\_Knob\_Changed) \wedge term\_Preselected\_Altitude' =$
$term\_Preselected\_Altitude)$

$Init\_Of\_term\_Preselected\_Altitude$

$aggr\_References$

$term\_Preselected\_Altitude = 31000$

$Def\_Of\_term\_Reference\_IAS$

$\Delta aggr\_References$
$\Delta aggr\_FCP\_Knobs$
$\Delta aggr\_Air\_Data$
$\Delta SYNC$
The_Event
$\Delta aggr\_Flight\_Modes$

$(mode\_Flight\_Director' \in FD\_ON \land mode\_Active\_Vertical \notin FLC \land$
$mode\_Active\_Vertical' \in FLC \land term\_SYNC' = FALS$
$\land\ term\_Reference\_IAS' = mon\_Indicated\_Airspeed')$
$\lor\ (mode\_Flight\_Director' \in FD\_ON \land mode\_Active\_Vertical' \in FLC \land term\_SYNC' = TRU$
$\land\ term\_Reference\_IAS' = mon\_Indicated\_Airspeed')$
$\lor\ (mode\_Flight\_Director \in FD\_ON$
$\land\ mode\_Active\_Vertical \notin FLC$
$\land\ event = Speed\_Knob\_Changed$
$\land\ mon\_Indicated\_Airspeed' = term\_Reference\_IAS' =$
$min\{512, max\{0, term\_Reference\_IAS + term\_Speed\_Knob\_Rotation'\}\})$
$\lor\ (mode\_Flight\_Director \in FD\_ON \land mode\_Active\_Vertical \in FLC$
$\land\ term\_SYNC' = FALS \land event = Speed\_Knob\_Changed$
$\land\ mon\_Indicated\_Airspeed' = term\_Reference\_IAS' =$
$min\{512, max\{0, term\_Reference\_IAS + term\_Speed\_Knob\_Rotation'\}\})$
$\lor\ (\neg\ ((mode\_Flight\_Director' \in FD\_ON \land mode\_Active\_Vertical \notin FLC \land$
$mode\_Active\_Vertical' \in FLC \land term\_SYNC' = FALS)$
$\lor\ (mode\_Flight\_Director' \in FD\_ON \land mode\_Active\_Vertical' \in FLC \land term\_SYNC' = TRU)$
$\lor\ (mode\_Flight\_Director \in FD\_ON$
$\land\ mode\_Active\_Vertical \notin FLC$
$\land\ event = Speed\_Knob\_Changed)$
$\lor\ (mode\_Flight\_Director \in FD\_ON \land mode\_Active\_Vertical \in FLC$
$\land\ term\_SYNC' = FALS \land event = Speed\_Knob\_Changed))$
$\land\ term\_Reference\_IAS' = term\_Reference\_IAS)$

$Def\_Of\_term\_Reference\_Heading$

$\Delta aggr\_References$
$\Delta aggr\_FCP\_Knobs$
$\Delta aggr\_Air\_Data$
$\Delta SYNC$
The_Event
$\Delta aggr\_Flight\_Modes$

$((\neg\ mode\_Active\_Lateral = ROLL\_HDG\_HOLD) \land$
$mode\_Active\_Lateral' = ROLL\_HDG\_HOLD \land term\_SYNC' = FALS$
$\land\ term\_Reference\_Heading' = mon\_Heading')$
$\lor\ (mode\_Active\_Lateral' = ROLL\_HDG\_HOLD \land term\_SYNC' = TRU$
$\land\ term\_Reference\_Heading' = mon\_Heading')$
$\lor\ (\neg\ (((\neg\ mode\_Active\_Lateral = ROLL\_HDG\_HOLD) \land$
$mode\_Active\_Lateral' = ROLL\_HDG\_HOLD \land term\_SYNC' = FALS)$
$\lor\ (mode\_Active\_Lateral' = ROLL\_HDG\_HOLD \land term\_SYNC' = TRU))$
$\land\ term\_Reference\_Heading' = term\_Reference\_Heading)$

$Def\_Of\_term\_Reference\_Altitude$

$\Delta aggr\_References$
$\Delta aggr\_FCP\_Knobs$
$\Delta aggr\_Air\_Data$
$\Delta SYNC$
The_Event
$\Delta aggr\_Flight\_Modes$

$((\neg\ mode\_Active\_Vertical = ALTHOLD) \land mode\_Active\_Vertical' = ALTHOLD \land$
$term\_Above\_Transition\_Altitude' = TRU \land term\_SYNC' = FALS$
$\land\ term\_Reference\_Altitude' = mon\_Pressure\_Altitude')$
$\lor\ (mode\_Active\_Vertical' = ALTHOLD \land term\_Above\_Transition\_Altitude' = TRU \land$
$term\_SYNC' = TRU$
$\land\ term\_Reference\_Altitude' = mon\_Pressure\_Altitude')$
$\lor\ ((\neg\ mode\_Active\_Vertical = ALTHOLD) \land mode\_Active\_Vertical' = ALTHOLD \land$
$term\_Above\_Transition\_Altitude' = FALS \land term\_SYNC' = FALS$
$\land\ term\_Reference\_Altitude' = mon\_Indicated\_Altitude')$
$\lor\ (mode\_Active\_Vertical' = ALTHOLD \land term\_Above\_Transition\_Altitude' = FALS$
$\land\ term\_SYNC' = TRU$
$\land\ term\_Reference\_Altitude' = mon\_Indicated\_Altitude')$
$\lor\ (\neg\ (((\neg\ mode\_Active\_Vertical = ALTHOLD) \land mode\_Active\_Vertical' = ALTHOLD \land$
$term\_Above\_Transition\_Altitude' = TRU \land term\_SYNC' = FALS)$
$\lor\ (mode\_Active\_Vertical' = ALTHOLD \land term\_Above\_Transition\_Altitude' = TRU \land$
$term\_SYNC' = TRU)$
$\lor\ ((\neg\ mode\_Active\_Vertical = ALTHOLD) \land mode\_Active\_Vertical' = ALTHOLD \land$
$term\_Above\_Transition\_Altitude' = FALS \land term\_SYNC' = FALS)$
$\lor\ (mode\_Active\_Vertical' = ALTHOLD \land term\_Above\_Transition\_Altitude' = FALS$
$\land\ term\_SYNC' = TRU))$
$\land\ term\_Reference\_Altitude' = term\_Reference\_Altitude)$

Def_Of_term_Reference_Pitch

$\Delta aggr\_References$
$\Delta aggr\_FCP\_Knobs$
$\Delta aggr\_Air\_Data$
$\Delta SYNC$
The_Event
$\Delta aggr\_Flight\_Modes$

$(mode\_Active\_Vertical \neq PITCH \wedge mode\_Active\_Vertical' = PITCH$
$\wedge\ term\_SYNC' = FALS$
$\wedge\ term\_Reference\_Pitch' = mon\_Pitch\_Angle')$
$\vee\ (mode\_Active\_Vertical' = PITCH \wedge term\_SYNC' = TRU$
$\wedge\ term\_Reference\_Pitch' = mon\_Pitch\_Angle')$
$\vee\ (mode\_Active\_Vertical = PITCH \wedge mode\_Active\_Vertical' = PITCH \wedge$
$event = VS\_Pitch\_Wheel\_Changed$
$\wedge\ term\_Reference\_Pitch' = max(\{-12, min(\{20,$
$term\_Reference\_Pitch + (term\_VS\_Pitch\_Wheel\_Rotation' \operatorname{div} 2)\})\}))$
$\vee\ (\neg\ ((mode\_Active\_Vertical \neq PITCH \wedge mode\_Active\_Vertical' = PITCH$
$\wedge\ term\_SYNC' = FALS)$
$\vee\ (mode\_Active\_Vertical' = PITCH \wedge term\_SYNC' = TRU)$
$\vee\ (mode\_Active\_Vertical = PITCH \wedge mode\_Active\_Vertical' = PITCH \wedge$
$event = VS\_Pitch\_Wheel\_Changed))$
$\wedge\ term\_Reference\_Pitch' = term\_Reference\_Pitch)$

Def_Of_term_Reference_Roll

$\Delta aggr\_References$
$\Delta aggr\_FCP\_Knobs$
$\Delta aggr\_Air\_Data$
$\Delta SYNC$
The_Event
$\Delta aggr\_Flight\_Modes$

$(mode\_Active\_Lateral \neq ROLL\_ROLL\_HOLD \wedge$
$mode\_Active\_Lateral' = ROLL\_ROLL\_HOLD \wedge term\_SYNC' = FALS$
$\wedge\ term\_Reference\_Roll' = max(\{-27, min(\{27, mon\_Roll\_Angle'\})\}))$
$\vee\ (mode\_Active\_Lateral' = ROLL\_ROLL\_HOLD \wedge term\_SYNC' = TRU$
$\wedge\ term\_Reference\_Roll' = max(\{-27, min(\{27, mon\_Roll\_Angle'\})\}))$
$\vee\ (\neg\ ((mode\_Active\_Lateral \neq ROLL\_ROLL\_HOLD \wedge$
$mode\_Active\_Lateral' = ROLL\_ROLL\_HOLD \wedge term\_SYNC' = FALS)$
$\vee\ (mode\_Active\_Lateral' = ROLL\_ROLL\_HOLD \wedge term\_SYNC' = TRU))$
$\wedge\ term\_Reference\_Roll' = term\_Reference\_Roll)$

$Def\_Of\_term\_Reference\_Vertical\_Speed$

$$\begin{array}{l}
\Delta aggr\_References \\
\Delta aggr\_FCP\_Knobs \\
\Delta aggr\_Air\_Data \\
\Delta SYNC \\
The\_Event \\
\Delta aggr\_Flight\_Modes \\
\hline
((\neg\,(mode\_Flight\_Director \in FD\_ON \land mode\_Active\_Vertical = VS)) \\
\land\,(mode\_Flight\_Director' \in FD\_ON \land mode\_Active\_Vertical' = VS) \\
\land\, term\_SYNC' = FALS \\
\land\, term\_Reference\_Vertical\_Speed' = mon\_Vertical\_Speed') \\
\lor\,(mode\_Flight\_Director' \in FD\_ON \land mode\_Active\_Vertical' = VS \\
\land\, term\_SYNC' = TRU \\
\land\, term\_Reference\_Vertical\_Speed' = mon\_Vertical\_Speed') \\
\lor\,(mode\_Flight\_Director \in FD\_ON \land mode\_Active\_Vertical = VS \land \\
term\_SYNC' = FALS \land event = VS\_Pitch\_Wheel\_Changed \\
\land\, term\_Reference\_Vertical\_Speed' = max(\{-50, min(\{50, \\
term\_Reference\_Vertical\_Speed + term\_VS\_Pitch\_Wheel\_Rotation\})\})) \\
\lor\,(\neg\,(((\neg\,(mode\_Flight\_Director \in FD\_ON \land mode\_Active\_Vertical = VS)) \\
\land\,(mode\_Flight\_Director' \in FD\_ON \land mode\_Active\_Vertical' = VS) \\
\land\, term\_SYNC' = FALS) \\
\lor\,(mode\_Flight\_Director' \in FD\_ON \land mode\_Active\_Vertical' = VS \\
\land\, term\_SYNC' = TRU) \\
\lor\,(mode\_Flight\_Director \in FD\_ON \land mode\_Active\_Vertical = VS \land \\
term\_SYNC' = FALS \land event = VS\_Pitch\_Wheel\_Changed)) \\
\land\, term\_Reference\_Vertical\_Speed' = term\_Reference\_Vertical\_Speed)
\end{array}$$

$Init\_Of\_term\_Reference\_IAS$

$$\begin{array}{l}
aggr\_References \\
\hline
term\_Reference\_IAS = 90
\end{array}$$

In the following schema, we multiply by 100 to get the desired precision.

$Def\_Of\_term\_Reference\_Mach$

$$\begin{array}{l}
aggr\_References \\
Speed\_Of\_Sound : \mathbb{N} \\
\hline
term\_Reference\_Mach = (100 * term\_Reference\_IAS) \operatorname{div} Speed\_Of\_Sound
\end{array}$$

$Init\_Of\_term\_Reference\_Vertical\_Speed$

$$\begin{array}{l}
aggr\_References \\
\hline
term\_Reference\_Vertical\_Speed = 0
\end{array}$$

A.9 REQ Relations for Some Annunciations

We have implemented the tolerances for some annunciations here.

This tolerance is 0.5 degrees; so we multiply by 10 to use integers.

REQ_Rel_for_con_Selected_Heading_Annunciation

Variables

$-5 \le 10 * con\_Selected\_Heading\_Annunciation - 10 * term\_Selected\_Heading \le 5$

This tolerance is for 20 feet.

REQ_Rel_for_con_Presel_Alt_Annunc

Variables

$-20 \le con\_Preselected\_Altitude\_Annunciation - term\_Preselected\_Altitude \le 20$

This tolerance is for 1 knot.

REQ_Rel_for_con_Reference_IAS_Annunciation

Variables

$-1 \le con\_Reference\_IAS\_Annunciation - term\_Reference\_IAS \le 1$

This tolerance is for 0.05 kft/min. Since the *ALTITUDE_RATE* type is measured in 0.001 kft/min increments, we multiply by 1000 to get integers.

REQ_Rel_for_con_Reference_VS_Annunciation

Variables

$-50 \le con\_Reference\_VS\_Annunciation - term\_Reference\_Vertical\_Speed \le 50$

A.10 More Term and Input Event Definitions

Def_Of_term_Overspeed

Overspeed

aggr_Air_Data

$term\_Overspeed = TRU \Leftrightarrow mode\_Overspeed = TOO\_FAST$

Def_Of_term_Above_Transition_Altitude

Variables

$term\_Above\_Transition\_Altitude = TRU$
$\Leftrightarrow (mon\_Pressure\_Altitude \le const\_Transition\_Altitude)$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Event\_Nav\_Source\_Changed}} \\
\hline
\Delta aggr\_Nav\_Source\_Mons \\
\Delta aggr\_Nav\_Source\_Data \\
The\_Event \\
\hline
event = Nav\_Source\_Changed \Leftrightarrow \\
(mon\_Selected\_Nav\_Source \neq mon\_Selected\_Nav\_Source' \lor \\
((mon\_Selected\_Nav\_Source\_Frequency(mon\_Selected\_Nav\_Source) \\
\neq mon\_Selected\_Nav\_Source\_Frequency(mon\_Selected\_Nav\_Source') \land \\
term\_Selected\_Nav\_Type \in \{VOR, LOC\}))) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Def\_Of\_term\_Selected\_Nav\_Type}} \\
\hline
aggr\_Nav\_Source\_Mons \\
aggr\_Nav\_Source\_Data \\
\hline
(mon\_Selected\_Nav\_Source \in \{FMS1, FMS2, FMS3\} \\
\land term\_Selected\_Nav\_Type = FMS) \\
\lor (mon\_Selected\_Nav\_Source \in \{VNR1, VNR2, VNR3, VNR4\} \land \\
mon\_Nav\_Source\_Signal\_Type(mon\_Selected\_Nav\_Source) = VOR \land \\
term\_Selected\_Nav\_Type = VOR) \\
\lor (mon\_Selected\_Nav\_Source \in \{VNR1, VNR2, VNR3, VNR4\} \land \\
mon\_Nav\_Source\_Signal\_Type(mon\_Selected\_Nav\_Source) = LOC \land \\
term\_Selected\_Nav\_Type = LOC) \\
\hline
\end{array}$$

We can use *term_SYNC* in the two definitions of input events below because of the definition of *term_SYNC*. These two definitions are a bit more convenient for use in Z/EVES.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Event\_SYNC\_On}} \\
\hline
The\_Event \\
\Delta SYNC \\
\hline
event = SYNC\_On \\
\Leftrightarrow (term\_SYNC = FALS \land term\_SYNC' = TRU) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Event\_SYNC\_Off}} \\
\hline
The\_Event \\
\Delta SYNC \\
\hline
event = SYNC\_Off \\
\Leftrightarrow (term\_SYNC = TRU \land term\_SYNC' = FALS) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Event\_AP\_Disconnect\_Bar\_Up}} \\
\hline
\Delta aggr\_FCP\_Switches \\
The\_Event \\
\hline
(event = AP\_Disconnect\_Bar\_Up) \Leftrightarrow \\
(mon\_AP\_Disconnect\_Bar = DOWN \land mon\_AP\_Disconnect\_Bar' = UP) \\
\hline
\end{array}$$

$Event\_AP\_Disconnect\_Bar\_Down$

$\Delta aggr\_FCP\_Switches$
The_Event

$(event = AP\_Disconnect\_Bar\_Down)$
$\Leftrightarrow (mon\_AP\_Disconnect\_Bar = UP \wedge mon\_AP\_Disconnect\_Bar' = DOWN)$

$Event\_Lateral\_NAV\_Track\_Cond\_Met$

$\Delta Lateral\_Terms$
The_Event

$(event = Lateral\_NAV\_Track\_Cond\_Met)$
$\Leftrightarrow (term\_Lateral\_NAV\_Track\_Cond\_Met = FALS \wedge$
$term\_Lateral\_NAV\_Track\_Cond\_Met' = TRU)$

$Event\_Lateral\_APPR\_Track\_Cond\_Met$

$\Delta Lateral\_Terms$
The_Event

$event = Lateral\_APPR\_Track\_Cond\_Met$
$\Leftrightarrow (term\_Lateral\_APPR\_Track\_Cond\_Met = FALS \wedge$
$term\_Lateral\_APPR\_Track\_Cond\_Met' = TRU)$

$Event\_Vert\_APPR\_Track\_Cond\_Met$

$\Delta Vertical\_Terms$
The_Event

$event = Vert\_APPR\_Track\_Cond\_Met \Leftrightarrow$
$term\_Vertical\_APPR\_Track\_Cond\_Met \neq TRU \wedge$
$term\_Vertical\_APPR\_Track\_Cond\_Met' = TRU$

$Event\_Gone\_Overspeed$

$\Delta aggr\_Air\_Data$
The_Event

$event = Gone\_Overspeed$
$\Leftrightarrow ((\neg (term\_Above\_Transition\_Altitude = FALS$
$\wedge mon\_Indicated\_Airspeed > term\_Vmo + 10) \wedge$
$(term\_Above\_Transition\_Altitude' = FALS$
$\wedge mon\_Indicated\_Airspeed' > term\_Vmo + 10)) \vee$
$(\neg (term\_Above\_Transition\_Altitude = TRU$
$\wedge mon\_Indicated\_Mach\_Number' > term\_Mmo + 3)$
$\wedge term\_Above\_Transition\_Altitude' = TRU$
$\wedge mon\_Indicated\_Mach\_Number' > term\_Mmo + 3))$

$Event\_Gone\_Normal$

$\Delta aggr\_Air\_Data$
The_Event

$event = Gone\_Normal$
$\Leftrightarrow ((\neg (term\_Above\_Transition\_Altitude = FALS \wedge$
$mon\_Indicated\_Airspeed \leq term\_Vmo) \wedge$
$term\_Above\_Transition\_Altitude' = FALS \wedge$
$mon\_Indicated\_Airspeed' \leq term\_Vmo) \vee$
$(\neg (term\_Above\_Transition\_Altitude = TRU \wedge$
$mon\_Indicated\_Mach\_Number \leq term\_Mmo) \wedge$
$term\_Above\_Transition\_Altitude' = TRU \wedge$
$mon\_Indicated\_Mach\_Number' \leq term\_Mmo))$

$Event\_ALTSEL\_TRACK\_Cond\_Met$

The_Event
$\Delta Altsel\_Terms$

$event = ALTSEL\_TRACK\_Cond\_Met$
$\Leftrightarrow (term\_ALTSEL\_Cond \neq ALTSEL\_COND\_TRACK \wedge$
$term\_ALTSEL\_Cond' = ALTSEL\_COND\_TRACK)$

$Event\_ALTSEL\_CAPTURE\_Cond\_Met$

The_Event
$\Delta Altsel\_Terms$

$event = ALTSEL\_CAPTURE\_Cond\_Met \Leftrightarrow$
$(term\_ALTSEL\_Cond \neq ALTSEL\_COND\_CAPTURE \wedge$
$term\_ALTSEL\_Cond' = ALTSEL\_COND\_CAPTURE)$

$Def\_Of\_term\_AP\_Engaged$

$Autopilot$

$term\_AP\_Engaged = TRU \Leftrightarrow mode\_Autopilot \in ENGAGED$

$REQ\_Rel\_for\_con\_AP\_Coupling$

$Autopilot$

$(((mode\_Autopilot = DISENGAGED\_WARNING)$
$\vee (mode\_Autopilot = DISENGAGED\_NORMAL) \vee$
$(mode\_Autopilot = ENGAGED\_SYNC))$
$\wedge con\_AP\_Coupling = MANUAL) \vee$
$(mode\_Autopilot = ENGAGED\_NORMAL$
$\wedge con\_AP\_Coupling = FGS)$

$\mathit{Event\_Land\_On\_Ground}$

$\Delta \mathit{aggr\_Air\_Data}$
$\mathit{The\_Event}$

$\mathit{event} = \mathit{Land\_On\_Ground} \Leftrightarrow \mathit{mon\_On\_Ground} = \mathit{FALS} \wedge \mathit{mon\_On\_Ground}' = \mathit{TRU}$

$\mathit{REQ\_Rel\_for\_con\_AP\_Disengage\_Warning}$

$\mathit{Autopilot}$

$(((\mathit{mode\_Autopilot} = \mathit{DISENGAGED\_NORMAL})$
$\vee (\mathit{mode\_Autopilot} = \mathit{ENGAGED\_NORMAL}) \vee$
$(\mathit{mode\_Autopilot} = \mathit{ENGAGED\_SYNC}))$
$\wedge\ \mathit{con\_AP\_Disengage\_Warning} = \mathit{OFF}) \vee$
$(\mathit{mode\_Autopilot} = \mathit{DISENGAGED\_WARNING}$
$\wedge\ \mathit{con\_AP\_Disengage\_Warning} = \mathit{ON})$

$\mathit{Def\_Of\_term\_Lateral\_NAV\_Track\_Cond\_Met}$

$\mathit{aggr\_Nav\_Source\_Data}$
$\mathit{aggr\_Nav\_Source\_Mons}$
$\mathit{Lateral\_Terms}$

$\mathit{term\_Lateral\_NAV\_Track\_Cond\_Met} = \mathit{TRU}$
$\Leftrightarrow (\mathit{term\_Selected\_Nav\_Type} \in \{\mathit{VOR}, \mathit{LOC}, \mathit{FMS}\}$
$\wedge\ \mathit{mon\_Selected\_Nav\_Source\_Status} = \mathit{VALID}$
$\wedge\ \mathit{term\_Within\_Lateral\_NAV\_Capture\_Window} = \mathit{TRU})$

$\mathit{Def\_Of\_term\_Lateral\_APPR\_Track\_Cond\_Met}$

$\mathit{aggr\_Nav\_Source\_Data}$
$\mathit{aggr\_Nav\_Source\_Mons}$
$\mathit{Lateral\_Terms}$

$\mathit{term\_Lateral\_APPR\_Track\_Cond\_Met} = \mathit{TRU}$
$\Leftrightarrow (\mathit{term\_Selected\_Nav\_Type} \in \{\mathit{LOC}, \mathit{FMS}\}$
$\wedge\ \mathit{mon\_Selected\_Nav\_Source\_Status} = \mathit{VALID}$
$\wedge\ \mathit{term\_Within\_Lateral\_APPR\_Capture\_Window} = \mathit{TRU})$

$\mathit{Def\_Of\_term\_Roll\_LE\_Threshold}$

$\mathit{aggr\_Air\_Data}$
$\mathit{Lateral\_Terms}$

$\mathit{term\_Roll\_LE\_Threshold} = \mathit{TRU} \Leftrightarrow$
$(\mathit{mon\_Roll\_Angle} \leq \mathit{const\_Roll\_Selection\_Threshold}$
$\wedge\ (-\mathit{mon\_Roll\_Angle} \leq \mathit{const\_Roll\_Selection\_Threshold}))$

$Def\_Of\_term\_Vertical\_APPR\_Track\_Cond\_Met$

$Vertical\_Terms$
$aggr\_Nav\_Source\_Data$
$aggr\_Nav\_Source\_Mons$

$term\_Vertical\_APPR\_Track\_Cond\_Met = TRU$
$\Leftrightarrow (term\_Selected\_Nav\_Type = LOC \wedge$
$mon\_Selected\_Nav\_Source\_Status = VALID \wedge mon\_Valid\_Glideslope = TRU$
$\wedge\ term\_Within\_Vertical\_APPR\_Capture\_Window = TRU)$

In *Transition_INMODE_Requirement*, we require that if an *INMODE* Boolean is *TRU* in a new state, then the corresponding mode machine was in the specified mode in the old state.

$Transition\_INMODE\_Requirement$

$\Delta INMODE\_Booleans$
$\Delta Variables$

$Duration\_INMODE\_AP\_Disengaged\_Warning\_gt\_ten\_sec' = TRU$
$\Rightarrow (mode\_Autopilot = DISENGAGED\_WARNING)$
$Duration\_INMODE\_Vert\_Appr\_Track\_gt\_const\_min\_armed\_period' = TRU$
$\Rightarrow (mode\_Vertical\_Approach = VERT\_APPR\_TRACK)$
$Duration\_INMODE\_NAV\_ARMED\_gt\_const\_min\_armed\_period' = TRU$
$\Rightarrow mode\_Active\_Lateral = NAV\_ARMED$
$Duration\_INMODE\_APPR\_ARMED\_gt\_const\_min\_armed\_period' = TRU$
$\Rightarrow mode\_Active\_Lateral = L\_APPR\_ARMED$
$Duration\_INMODE\_ALTSEL\_ARMED\_gt\_const\_min\_armed\_period' = TRU$
$\Rightarrow mode\_Altitude\_Select = ALTSEL\_ARMED$
$Duration\_INMODE\_ALTSEL\_CAPT\_gt\_const\_min\_armed\_period' = TRU$
$\Rightarrow mode\_Altitude\_Select = ALTSEL\_CAPTURE$

A.11 Lamp Annunciations

The following schemas describe the conditions under which these lamps are lit:
con_HDG_Switch_Lamp, *con_NAV_Switch_Lamp*,
con_APPR_Switch_Lamp, *con_ALT_Switch_Lamp*,
con_VS_Switch_Lamp, *con_FLC_Switch_Lamp*, and
con_AP_Engaged_Switch_Lamp.

$REQ\_Rel\_for\_con\_HDG\_Switch\_Lamp$

$aggr\_Flight\_Modes$
$aggr\_FCP\_Lamps$

$(mode\_Flight\_Director = FD\_OFF \wedge con\_HDG\_Switch\_Lamp = UNLIT) \vee$
$(mode\_Flight\_Director \in FD\_ON \wedge (mode\_Active\_Lateral \neq HDG$
$\wedge\ con\_HDG\_Switch\_Lamp = UNLIT) \vee$
$mode\_Active\_Lateral = HDG \wedge con\_HDG\_Switch\_Lamp = LIT)$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{REQ\_Rel\_for\_con\_NAV\_Switch\_Lamp}} \\
aggr\_Flight\_Modes \\
aggr\_FCP\_Lamps \\
\hline
(mode\_Flight\_Director = FD\_OFF \land con\_NAV\_Switch\_Lamp = UNLIT) \lor \\
(mode\_Flight\_Director \in FD\_ON \land (mode\_Active\_Lateral \notin NAV \\
\land con\_NAV\_Switch\_Lamp = UNLIT) \lor \\
mode\_Active\_Lateral \in NAV \land con\_NAV\_Switch\_Lamp = LIT) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{REQ\_Rel\_for\_con\_APPR\_Switch\_Lamp}} \\
aggr\_Flight\_Modes \\
aggr\_FCP\_Lamps \\
\hline
(mode\_Flight\_Director = FD\_OFF \land con\_APPR\_Switch\_Lamp = UNLIT) \lor \\
(mode\_Flight\_Director \in FD\_ON \land (mode\_Active\_Lateral \notin L\_APPR \\
\land con\_APPR\_Switch\_Lamp = UNLIT) \lor \\
mode\_Active\_Lateral \in L\_APPR \land con\_APPR\_Switch\_Lamp = LIT) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{REQ\_Rel\_for\_con\_ALT\_Switch\_Lamp}} \\
aggr\_Flight\_Modes \\
aggr\_FCP\_Lamps \\
\hline
(mode\_Flight\_Director = FD\_OFF \land con\_ALT\_Switch\_Lamp = UNLIT) \lor \\
(mode\_Flight\_Director \in FD\_ON \land (mode\_Active\_Vertical \neq ALTHOLD \\
\land con\_ALT\_Switch\_Lamp = UNLIT) \lor \\
mode\_Active\_Vertical = ALTHOLD \land con\_ALT\_Switch\_Lamp = LIT) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{REQ\_Rel\_for\_con\_VS\_Switch\_Lamp}} \\
aggr\_Flight\_Modes \\
aggr\_FCP\_Lamps \\
\hline
(mode\_Flight\_Director = FD\_OFF \land con\_VS\_Switch\_Lamp = UNLIT) \lor \\
(mode\_Flight\_Director \in FD\_ON \land (mode\_Active\_Vertical \neq VS \\
\land con\_VS\_Switch\_Lamp = UNLIT) \lor \\
mode\_Active\_Vertical = VS \land con\_VS\_Switch\_Lamp = LIT) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{REQ\_Rel\_for\_con\_FLC\_Switch\_Lamp}} \\
aggr\_Flight\_Modes \\
aggr\_FCP\_Lamps \\
\hline
(mode\_Flight\_Director = FD\_OFF \land con\_FLC\_Switch\_Lamp = UNLIT) \lor \\
(mode\_Flight\_Director \in FD\_ON \land (mode\_Active\_Vertical \notin FLC \\
\land con\_FLC\_Switch\_Lamp = UNLIT) \lor \\
mode\_Active\_Vertical \in FLC \land con\_FLC\_Switch\_Lamp = LIT) \\
\hline
\end{array}$$

$REQ\_Rel\_for\_con\_AP\_Engaged\_Switch\_Lamp$

$aggr\_Flight\_Modes$
$aggr\_FCP\_Lamps$
$Autopilot$

$(mode\_Autopilot \in DISENGAGED \wedge con\_AP\_Engaged\_Switch\_Lamp = UNLIT) \vee$
$(mode\_Autopilot \in ENGAGED \wedge con\_AP\_Engaged\_Switch\_Lamp = LIT)$

A.12 Collections of Terms and Events

In the schema Conditioned_Terms, we collect the definitions of the term variables that are defined by formulas or condition tables. These can be included as invariants in the definition of *State*.

$Conditioned\_Terms$

$Def\_Of\_term\_SYNC$
$Def\_Of\_term\_Reference\_Mach$
$Def\_Of\_term\_Overspeed$
$Def\_Of\_term\_Above\_Transition\_Altitude$
$Def\_Of\_term\_Selected\_Nav\_Type$
$Def\_Of\_term\_AP\_Engaged$
$Def\_Of\_term\_Lateral\_NAV\_Track\_Cond\_Met$
$Def\_Of\_term\_Lateral\_APPR\_Track\_Cond\_Met$
$Def\_Of\_term\_Roll\_LE\_Threshold$
$Def\_Of\_term\_Vertical\_APPR\_Track\_Cond\_Met$

In the schema *REQ_Relations*, we collect the REQ relations imposed on controlled variables.

$REQ\_Relations$

$REQ\_Rel\_for\_con\_Selected\_Heading\_Annunciation$
$REQ\_Rel\_for\_con\_Presel\_Alt\_Annunc$
$REQ\_Rel\_for\_con\_Reference\_IAS\_Annunciation$
$REQ\_Rel\_for\_con\_Reference\_VS\_Annunciation$
$REQ\_Rel\_for\_con\_AP\_Coupling$
$REQ\_Rel\_for\_con\_AP\_Disengage\_Warning$
$REQ\_Rel\_for\_con\_HDG\_Switch\_Lamp$
$REQ\_Rel\_for\_con\_NAV\_Switch\_Lamp$
$REQ\_Rel\_for\_con\_APPR\_Switch\_Lamp$
$REQ\_Rel\_for\_con\_ALT\_Switch\_Lamp$
$REQ\_Rel\_for\_con\_VS\_Switch\_Lamp$
$REQ\_Rel\_for\_con\_FLC\_Switch\_Lamp$
$REQ\_Rel\_for\_con\_AP\_Engaged\_Switch\_Lamp$

In the schema *Events_And_Event_Terms*, we collect term variables that are defined by event tables, as well as the event definitions.

```
— Events_And_Event_Terms ————————————————————————
| Event_HDG_Switch_Pressed
| Event_NAV_Switch_Pressed
| Event_APPR_Switch_Pressed
| Event_ALT_Switch_Pressed
| Event_FLC_Switch_Pressed
| Def_Of_term_VS_Pitch_Wheel_Rotation
| Event_VS_Pitch_Wheel_Changed
| Def_Of_term_ALT_Knob_Rotation
| Event_ALT_Knob_Changed
| Def_Of_term_Speed_Knob_Rotation
| Event_Speed_Knob_Changed
| Def_Of_term_HDG_Knob_Rotation
| Event_HDG_Knob_Changed
| Event_AP_Engage_Switch_Pressed
| Event_FD_Pressed
| Event_AP_Disengage_Pressed
| Event_GA_Pressed
| Def_Of_term_Selected_Heading
| Def_Of_term_Preselected_Altitude
| Def_Of_term_Reference_Heading
| Def_Of_term_Reference_Pitch
| Def_Of_term_Reference_Roll
| Def_Of_term_Reference_Vertical_Speed
| Event_Nav_Source_Changed
| Transition_INMODE_Requirement
| Event_VS_Switch_Pressed
| Def_Of_term_Reference_IAS
| Def_Of_term_Reference_Altitude
| Event_SYNC_On
| Event_SYNC_Off
| Event_AP_Disconnect_Bar_Up
| Event_AP_Disconnect_Bar_Down
| Event_Land_On_Ground
| Event_Lateral_NAV_Track_Cond_Met
| Event_Lateral_APPR_Track_Cond_Met
| Event_Vert_APPR_Track_Cond_Met
| Event_Gone_Overspeed
| Event_Gone_Normal
| Event_ALTSEL_TRACK_Cond_Met
| Event_ALTSEL_CAPTURE_Cond_Met
———————————————————————————————————————————————————
```

Here we declare the state of the system, which consists of the values of the variables, as well as invariants that define some of the term variables and controlled variables.

State

$\mathit{Variables}$

$\mathit{Conditioned\_Terms}$

$\mathit{REQ\_Relations}$

Here we declare the transition of the system, which consists of an old state, a new state, an input event, and invariants that define some of the term variables.

$\mathit{Transition}$

$\Delta \mathit{State}$

$\mathit{The\_Event}$

$\mathit{Events\_And\_Event\_Terms}$

The following schema is the mode machine initialization for *mode_Flight_Director*.

$\mathit{Init\_Of\_mode\_Flight\_Director}$

State

$\mathit{mode\_Flight\_Director} = \mathit{FD\_OFF}$

The following mode machine initializations for
mode_Active_Lateral,
mode_Active_Vertical,
mode_Altitude_Select, and
mode_Vertical_Approach
are different from the CoRE initializations because we have simulated the sustaining conditions of these mode machines using *NOT_IN_MODE* modes. The CoRE initializations now appear as transitions in the respective tables.

$\mathit{Init\_Of\_mode\_Active\_Lateral}$

State

$\mathit{mode\_Active\_Lateral} = \mathit{LATERAL\_NOT\_IN\_MODE}$

$\mathit{Init\_Of\_mode\_Active\_Vertical}$

State

$\mathit{mode\_Active\_Vertical} = \mathit{VERTICAL\_NOT\_IN\_MODE}$

$\mathit{Init\_Of\_mode\_Altitude\_Select}$

State

$\mathit{mode\_Altitude\_Select} = \mathit{ALTSEL\_NOT\_IN\_MODE}$

$\mathit{Init\_Of\_mode\_Vertical\_Approach}$

State

$\mathit{mode\_Vertical\_Approach} = \mathit{VERT\_APPR\_NOT\_IN\_MODE}$

$$\begin{array}{|l}
\textit{InitState} \\
\hline
\textit{State} \\
\textit{Init\_Of\_Overspeed} \\
\textit{Init\_Of\_Autopilot} \\
\textit{Init\_Of\_mode\_Flight\_Director} \\
\textit{Init\_Of\_term\_Selected\_Heading} \\
\textit{Init\_Of\_term\_Preselected\_Altitude} \\
\textit{Init\_Of\_term\_Reference\_IAS} \\
\textit{Init\_Of\_term\_Reference\_Vertical\_Speed} \\
\textit{Init\_Of\_mode\_Active\_Lateral} \\
\textit{Init\_Of\_mode\_Active\_Vertical} \\
\textit{Init\_Of\_mode\_Altitude\_Select} \\
\textit{Init\_Of\_mode\_Vertical\_Approach} \\
\hline
\end{array}$$

A.13 Invariants

This invariant arises from the CoRE sustaining condition imposed on four of the mode machines. We have augmented the transition tables of these machines to preserve the invariant.

$$\begin{array}{|l}
\textit{FD\_Invariant} \\
\hline
\textit{State} \\
\hline
\textit{mode\_Flight\_Director} = \textit{FD\_OFF} \Rightarrow \\
(\textit{mode\_Active\_Lateral} = \textit{LATERAL\_NOT\_IN\_MODE} \land \\
\textit{mode\_Active\_Vertical} = \textit{VERTICAL\_NOT\_IN\_MODE} \land \\
\textit{mode\_Altitude\_Select} = \textit{ALTSEL\_NOT\_IN\_MODE} \land \\
\textit{mode\_Vertical\_Approach} = \textit{VERT\_APPR\_NOT\_IN\_MODE}) \\
\hline
\end{array}$$

The following twelve invariants are translated from the CoRE specification.

$$\begin{array}{|l}
\textit{Invariant\_One} \\
\hline
\textit{State} \\
\hline
\textit{mode\_Active\_Lateral} = \textit{L\_GA} \Rightarrow \textit{mode\_Autopilot} \in \textit{DISENGAGED} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\textit{Invariant\_Two} \\
\hline
\textit{State} \\
\hline
\textit{mode\_Active\_Vertical} = \textit{V\_GA} \Rightarrow \textit{mode\_Autopilot} \in \textit{DISENGAGED} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\textit{Invariant\_Three} \\
\hline
\textit{State} \\
\hline
\textit{term\_AP\_Engaged} = \textit{TRU} \Rightarrow \textit{mode\_Flight\_Director} \in \textit{FD\_ON} \\
\hline
\end{array}$$

Invariant_Four

State

$$\begin{array}{l}(mode\_Flight\_Director \in FD\_ON \wedge \\ mode\_Active\_Lateral \in ROLL \wedge \\ mon\_On\_Ground = TRU) \Rightarrow mode\_Active\_Lateral = ROLL\_HDG\_HOLD\end{array}$$

Invariant_Five

State

$$\begin{array}{l}(mode\_Flight\_Director \in FD\_ON \wedge mode\_Active\_Vertical = V\_GA) \\ \Rightarrow mode\_Active\_Lateral = L\_GA\end{array}$$

Invariant_Six

State

$$\begin{array}{l}(mode\_Flight\_Director \in FD\_ON \wedge \\ mode\_Active\_Lateral = NAV\_TRACK) \\ \Rightarrow term\_Selected\_Nav\_Type \in \{VOR, LOC, FMS\}\end{array}$$

Invariant_Seven

State

$$\begin{array}{l}mode\_Flight\_Director \in FD\_ON \Rightarrow \\ (mode\_Altitude\_Select = ALTSEL\_CLEARED \\ \Leftrightarrow mode\_Active\_Vertical \in \{V\_APPR, V\_GA, ALTHOLD\})\end{array}$$

Invariant_Eight

State

$$\begin{array}{l}mode\_Flight\_Director \in FD\_ON \Rightarrow \\ (mode\_Altitude\_Select \in ALTSEL\_ACTIVE \\ \Leftrightarrow mode\_Active\_Vertical \in \{V\_APPR, V\_GA, ALTHOLD\})\end{array}$$

Invariant_Nine

State

$$\begin{array}{l}mode\_Flight\_Director \in FD\_ON \Rightarrow (mode\_Altitude\_Select \in ALTSEL\_ACTIVE \\ \Leftrightarrow mode\_Active\_Vertical = ALTSEL)\end{array}$$

Invariant_Ten

State

$$\begin{array}{l}mode\_Flight\_Director \in FD\_ON \Rightarrow \\ (mode\_Vertical\_Approach = VERT\_APPR\_TRACK \\ \Leftrightarrow mode\_Active\_Vertical = V\_APPR)\end{array}$$

$Invariant\_Eleven$

$State$

$mode\_Flight\_Director \in FD\_ON$
$\Rightarrow (term\_Overspeed = TRU$
$\Leftrightarrow mode\_Active\_Vertical \in \{ALTSEL, ALTHOLD, V\_APPR, FLC\_OVERSPEED\})$

$Invariant\_Twelve$

$State$

$mode\_Flight\_Director \in FD\_ON$
$\Rightarrow (mode\_Active\_Lateral = L\_GA \Rightarrow mode\_Active\_Vertical = V\_GA)$

$Legal\_State$

$State$
$FD\_Invariant$
$Invariant\_One$
$Invariant\_Two$
$Invariant\_Three$
$Invariant\_Four$
$Invariant\_Five$
$Invariant\_Six$
$Invariant\_Seven$
$Invariant\_Eight$
$Invariant\_Nine$
$Invariant\_Ten$
$Invariant\_Eleven$
$Invariant\_Twelve$

A.14 Transition Tables

$mode\_Overspeed\_Transition\_One$

$Transition$

$(mode\_Overspeed = SPEED\_OK \wedge$
$(\neg (term\_Above\_Transition\_Altitude = FALS \wedge mon\_Indicated\_Airspeed > term\_Vmo + 10)$
$\wedge (term\_Above\_Transition\_Altitude' = FALS \wedge mon\_Indicated\_Airspeed' > term\_Vmo + 10))$
$\wedge mode\_Overspeed' = TOO\_FAST)$

$mode\_Overspeed\_Transition\_Two$

$Transition$

$(mode\_Overspeed = SPEED\_OK \wedge$
$(\neg (term\_Above\_Transition\_Altitude = TRU \wedge mon\_Indicated\_Mach\_Number' > term\_Mmo + 3)$
$\wedge term\_Above\_Transition\_Altitude' = TRU \wedge mon\_Indicated\_Mach\_Number' > term\_Mmo + 3)$
$\wedge mode\_Overspeed' = TOO\_FAST)$

mode_Overspeed_Transition_Three

Transition

$(mode\_Overspeed = TOO\_FAST \land$
$(\neg\,(term\_Above\_Transition\_Altitude = FALS \land mon\_Indicated\_Airspeed \leq term\_Vmo)$
$\land\ term\_Above\_Transition\_Altitude' = FALS \land mon\_Indicated\_Airspeed' \leq term\_Vmo)$
$\land\ mode\_Overspeed' = SPEED\_OK)$

mode_Overspeed_Transition_Four

Transition

$(mode\_Overspeed = TOO\_FAST$
$\land\ (\neg\,(term\_Above\_Transition\_Altitude = TRU \land mon\_Indicated\_Mach\_Number \leq term\_Mmo)$
$\land\ term\_Above\_Transition\_Altitude' = TRU \land mon\_Indicated\_Mach\_Number' \leq term\_Mmo)$
$\land\ mode\_Overspeed' = SPEED\_OK)$

mode_Overspeed_Transition_Five

Transition

$(\neg\,((mode\_Overspeed = SPEED\_OK \land$
$(\neg\,(term\_Above\_Transition\_Altitude = FALS \land mon\_Indicated\_Airspeed > term\_Vmo + 10)$
$\land\ (term\_Above\_Transition\_Altitude' = FALS \land mon\_Indicated\_Airspeed' > term\_Vmo + 10)))$
$\lor\ (mode\_Overspeed = SPEED\_OK \land$
$(\neg\,(term\_Above\_Transition\_Altitude = TRU \land mon\_Indicated\_Mach\_Number' > term\_Mmo + 3)$
$\land\ term\_Above\_Transition\_Altitude' = TRU \land mon\_Indicated\_Mach\_Number' > term\_Mmo + 3))$
$\lor\ (mode\_Overspeed = TOO\_FAST \land$
$(\neg\,(term\_Above\_Transition\_Altitude = FALS \land mon\_Indicated\_Airspeed \leq term\_Vmo)$
$\land\ term\_Above\_Transition\_Altitude' = FALS \land mon\_Indicated\_Airspeed' \leq term\_Vmo))$
$\lor\ (mode\_Overspeed = TOO\_FAST$
$\land\ (\neg\,(term\_Above\_Transition\_Altitude = TRU \land mon\_Indicated\_Mach\_Number \leq term\_Mmo)$
$\land\ term\_Above\_Transition\_Altitude' = TRU \land mon\_Indicated\_Mach\_Number' \leq term\_Mmo)))$
$\land\ mode\_Overspeed' = mode\_Overspeed)$

$mode\_Overspeed\_Transition\_Table \mathrel{\widehat=}$
$mode\_Overspeed\_Transition\_One \lor$
$mode\_Overspeed\_Transition\_Two \lor$
$mode\_Overspeed\_Transition\_Three \lor$
$mode\_Overspeed\_Transition\_Four \lor$
$mode\_Overspeed\_Transition\_Five$

mode_Autopilot_Transition_One

Transition

$((event = AP\_Engage\_Switch\_Pressed \land mon\_AP\_Disconnect\_Bar = UP$
$\land\ mode\_Autopilot \in DISENGAGED)$
$\land\ mode\_Autopilot' = ENGAGED\_NORMAL)$

$mode\_Autopilot\_Transition\_Two$

$Transition$

$((event = AP\_Disengage\_Pressed \land mode\_Autopilot \in ENGAGED)$
$\land\ mode\_Autopilot' = DISENGAGED\_NORMAL)$

$mode\_Autopilot\_Transition\_Three$

$Transition$

$((mode\_Autopilot \in ENGAGED \land mon\_AP\_Disconnect\_Bar \neq DOWN$
$\land\ mon\_AP\_Disconnect\_Bar' = DOWN)$
$\land\ mode\_Autopilot = DISENGAGED\_NORMAL)$

$mode\_Autopilot\_Transition\_Four$

$Transition$

$((mode\_Autopilot \in ENGAGED \land$
$((mode\_Active\_Lateral \neq L\_GA \land mode\_Active\_Lateral' = L\_GA) \lor$
$(mode\_Active\_Vertical \neq V\_GA \land mode\_Active\_Vertical' = V\_GA)))$
$\land\ mode\_Autopilot' = DISENGAGED\_WARNING)$

$mode\_Autopilot\_Transition\_Five$

$Transition$

$((mode\_Autopilot = ENGAGED\_NORMAL \land$
$term\_SYNC = FALS \land term\_SYNC' = TRU)$
$\land\ mode\_Autopilot' = ENGAGED\_SYNC)$

$mode\_Autopilot\_Transition\_Six$

$Transition$

$((mode\_Autopilot = ENGAGED\_SYNC \land$
$term\_SYNC' = FALS \land term\_SYNC = TRU)$
$\land\ mode\_Autopilot' = ENGAGED\_NORMAL)$

$mode\_Autopilot\_Transition\_Seven$

$Transition$

$((mode\_Autopilot = DISENGAGED\_WARNING \land$
$Duration\_INMODE\_AP\_Disengaged\_Warning\_gt\_ten\_sec = FALS \land$
$Duration\_INMODE\_AP\_Disengaged\_Warning\_gt\_ten\_sec' = TRU)$
$\land\ mode\_Autopilot' = DISENGAGED\_NORMAL)$

$mode\_Autopilot\_Transition\_Eight$

$Transition$

$(\neg (((event = AP\_Engage\_Switch\_Pressed \wedge mon\_AP\_Disconnect\_Bar = UP$
$\wedge\ mode\_Autopilot \in DISENGAGED))$
$\vee\ ((event = AP\_Disengage\_Pressed \wedge mode\_Autopilot \in ENGAGED))$
$\vee\ ((mode\_Autopilot \in ENGAGED \wedge mon\_AP\_Disconnect\_Bar \neq DOWN$
$\wedge\ mon\_AP\_Disconnect\_Bar' = DOWN))$
$\vee\ ((mode\_Autopilot \in ENGAGED \wedge$
$((mode\_Active\_Lateral \neq L\_GA \wedge mode\_Active\_Lateral' = L\_GA) \vee$
$(mode\_Active\_Vertical \neq V\_GA \wedge mode\_Active\_Vertical' = V\_GA))))$
$\vee\ ((mode\_Autopilot = ENGAGED\_NORMAL \wedge$
$term\_SYNC = FALS \wedge term\_SYNC' = TRU))$
$\vee\ ((mode\_Autopilot = ENGAGED\_SYNC \wedge$
$term\_SYNC' = FALS \wedge term\_SYNC = TRU))$
$\vee\ ((mode\_Autopilot = DISENGAGED\_WARNING \wedge$
$Duration\_INMODE\_AP\_Disengaged\_Warning\_gt\_ten\_sec = FALS \wedge$
$Duration\_INMODE\_AP\_Disengaged\_Warning\_gt\_ten\_sec' = TRU)))$
$\wedge\ mode\_Autopilot' = mode\_Autopilot)$

$mode\_Autopilot\_Transition\_Table \mathrel{\widehat=}$
$mode\_Autopilot\_Transition\_One \vee$
$mode\_Autopilot\_Transition\_Two \vee$
$mode\_Autopilot\_Transition\_Three \vee$
$mode\_Autopilot\_Transition\_Four \vee$
$mode\_Autopilot\_Transition\_Five \vee$
$mode\_Autopilot\_Transition\_Six \vee$
$mode\_Autopilot\_Transition\_Seven \vee$
$mode\_Autopilot\_Transition\_Eight$

$mode\_Flight\_Director\_Transition\_One$

$Transition$

$((event \in Flight\_Mode\_Requested \wedge mode\_Flight\_Director = FD\_OFF)$
$\wedge\ mode\_Flight\_Director' = FD\_ON\_CUES)$

$mode\_Flight\_Director\_Transition\_Two$

$Transition$

$((event = Gone\_Overspeed \wedge mode\_Flight\_Director = FD\_OFF)$
$\wedge\ mode\_Flight\_Director' = FD\_ON\_CUES)$

$mode\_Flight\_Director\_Transition\_Three$

$Transition$

$((event = AP\_Engage\_Switch\_Pressed \land$
$(mode\_Autopilot \notin ENGAGED \land mode\_Autopilot' \in ENGAGED)$
$\land\ mode\_Flight\_Director = FD\_OFF)$
$\land\ mode\_Flight\_Director' = FD\_ON\_CUES)$

$mode\_Flight\_Director\_Transition\_Four$

$Transition$

$((event = FD\_Pressed \land mode\_Flight\_Director = FD\_OFF)$
$\land\ mode\_Flight\_Director' = FD\_ON\_CUES)$

$mode\_Flight\_Director\_Transition\_Five$

$Transition$

$((event = FD\_Pressed \land$
$mode\_Flight\_Director \in FD\_ON \land term\_Overspeed = FALS$
$\land\ term\_AP\_Engaged = FALS)$
$\land\ mode\_Flight\_Director' = FD\_OFF)$

$mode\_Flight\_Director\_Transition\_Six$

$Transition$

$(mode\_Flight\_Director = FD\_ON\_NO\_CUES \land$
$event = FD\_Pressed \land$
$(term\_AP\_Engaged = TRU \lor term\_Overspeed = TRU)$
$\land\ mode\_Flight\_Director = FD\_ON\_CUES)$

$mode\_Flight\_Director\_Transition\_Seven$

$Transition$

$((mode\_Flight\_Director = FD\_ON\_NO\_CUES \land event = Gone\_Overspeed)$
$\land\ mode\_Flight\_Director = FD\_ON\_CUES)$

$mode\_Flight\_Director\_Transition\_Eight$

$Transition$

$(mode\_Flight\_Director = FD\_ON\_CUES \land$
$event = FD\_Pressed \land$
$(term\_AP\_Engaged = TRU \lor term\_Overspeed = TRU)$
$\land\ mode\_Flight\_Director = FD\_ON\_NO\_CUES)$

$mode\_Flight\_Director\_Transition\_Nine$

$Transition$

$$
\begin{array}{l}
(\neg\ (((event \in Flight\_Mode\_Requested \wedge mode\_Flight\_Director = FD\_OFF)) \\
\vee\ ((event = Gone\_Overspeed \wedge mode\_Flight\_Director = FD\_OFF)) \\
\vee\ ((event = AP\_Engage\_Switch\_Pressed \wedge \\
(mode\_Autopilot \notin ENGAGED \wedge mode\_Autopilot' \in ENGAGED) \\
\wedge\ mode\_Flight\_Director = FD\_OFF)) \\
\vee\ ((event = FD\_Pressed \wedge mode\_Flight\_Director = FD\_OFF)) \\
\vee\ ((event = FD\_Pressed \wedge \\
mode\_Flight\_Director \in FD\_ON \wedge term\_Overspeed = FALS \\
\wedge\ term\_AP\_Engaged = FALS)) \\
\vee\ (mode\_Flight\_Director = FD\_ON\_NO\_CUES \wedge \\
event = FD\_Pressed \wedge \\
(term\_AP\_Engaged = TRU \vee term\_Overspeed = TRU)) \\
\vee\ ((mode\_Flight\_Director = FD\_ON\_NO\_CUES \wedge event = Gone\_Overspeed)) \\
\vee\ (mode\_Flight\_Director = FD\_ON\_CUES \wedge \\
event = FD\_Pressed \wedge \\
(term\_AP\_Engaged = TRU \vee term\_Overspeed = TRU))) \\
\wedge\ mode\_Flight\_Director' = mode\_Flight\_Director)
\end{array}
$$

$$
\begin{array}{l}
mode\_Flight\_Director\_Transition\_Table \mathrel{\widehat=} \\
mode\_Flight\_Director\_Transition\_One \vee \\
mode\_Flight\_Director\_Transition\_Two \vee \\
mode\_Flight\_Director\_Transition\_Three \vee \\
mode\_Flight\_Director\_Transition\_Four \vee \\
mode\_Flight\_Director\_Transition\_Five \vee \\
mode\_Flight\_Director\_Transition\_Six \vee \\
mode\_Flight\_Director\_Transition\_Seven \vee \\
mode\_Flight\_Director\_Transition\_Eight \vee \\
mode\_Flight\_Director\_Transition\_Nine
\end{array}
$$

$mode\_Active\_Lateral\_Transition\_One$

$Transition$

$$
\begin{array}{l}
(mode\_Flight\_Director \neq FD\_OFF \wedge mode\_Flight\_Director' = FD\_OFF \\
\wedge\ mode\_Active\_Lateral' = LATERAL\_NOT\_IN\_MODE)
\end{array}
$$

$mode\_Active\_Lateral\_Transition\_Two$

$Transition$

$$
\begin{array}{l}
(mode\_Flight\_Director = FD\_OFF \wedge mode\_Flight\_Director' \neq FD\_OFF \wedge \\
event \notin Lateral\_Mode\_Requested \wedge event \neq GA\_Pressed \\
\wedge\ mode\_Active\_Lateral' \in ROLL)
\end{array}
$$

$mode\_Active\_Lateral\_Transition\_Three$

$Transition$

$((event = HDG\_Switch\_Pressed \land mode\_Active\_Lateral = HDG)$
$\land\ mode\_Active\_Lateral' \in ROLL)$

$mode\_Active\_Lateral\_Transition\_Four$

$Transition$

$((event = NAV\_Switch\_Pressed \land mode\_Active\_Lateral \in NAV)$
$\land\ mode\_Active\_Lateral' \in ROLL)$

$mode\_Active\_Lateral\_Transition\_Five$

$Transition$

$((event = Nav\_Source\_Changed \land mode\_Active\_Lateral \in NAV)$
$\land\ mode\_Active\_Lateral' \in ROLL)$

$mode\_Active\_Lateral\_Transition\_Six$

$Transition$

$((event = APPR\_Switch\_Pressed \land mode\_Active\_Lateral \in L\_APPR)$
$\land\ mode\_Active\_Lateral' \in ROLL)$

$mode\_Active\_Lateral\_Transition\_Seven$

$Transition$

$((event = Nav\_Source\_Changed \land mode\_Active\_Lateral \in L\_APPR)$
$\land\ mode\_Active\_Lateral' \in ROLL)$

$mode\_Active\_Lateral\_Transition\_Eight$

$Transition$

$((term\_AP\_Engaged \neq TRU \land term\_AP\_Engaged' = TRU$
$\land\ mode\_Active\_Lateral = L\_GA)$
$\land\ mode\_Active\_Lateral' \in ROLL)$

$mode\_Active\_Lateral\_Transition\_Nine$

$Transition$

$((term\_SYNC = FALS \land term\_SYNC' = TRU \land mode\_Active\_Lateral = L\_GA)$
$\land\ mode\_Active\_Lateral' \in ROLL)$

$mode\_Active\_Lateral\_Transition\_Ten$

$Transition$

$((mode\_Active\_Vertical = V\_GA \land mode\_Active\_Vertical' \neq V\_GA \land$
$mode\_Active\_Lateral = L\_GA \land event \in Vertical\_Events \setminus Lateral\_Events)$
$\land\ mode\_Active\_Lateral' \in ROLL)$

$mode\_Active\_Lateral\_Transition\_Eleven$

$Transition$

$((event = HDG\_Switch\_Pressed \land mode\_Active\_Lateral \neq HDG)$
$\land\ mode\_Active\_Lateral' = HDG)$

$mode\_Active\_Lateral\_Transition\_Twelve$

$Transition$

$((event = NAV\_Switch\_Pressed \land mode\_Active\_Lateral' \notin NAV)$
$\land\ mode\_Active\_Lateral' = NAV\_ARMED)$

$mode\_Active\_Lateral\_Transition\_Thirteen$

$Transition$

$((event = APPR\_Switch\_Pressed \land mode\_Active\_Lateral \notin L\_APPR)$
$\land\ mode\_Active\_Lateral' = L\_APPR\_ARMED)$

$mode\_Active\_Lateral\_Transition\_Fourteen$

$Transition$

$((event = GA\_Pressed \land mode\_Active\_Lateral \neq L\_GA)$
$\land\ mode\_Active\_Lateral' = L\_GA)$

$mode\_Active\_Lateral\_Transition\_Fifteen$

$Transition$

$(mode\_Active\_Lateral \notin ROLL \land mode\_Active\_Lateral' \in ROLL$
$\land\ (term\_Roll\_LE\_Threshold = TRU \lor mon\_On\_Ground = TRU)$
$\land\ mode\_Active\_Lateral' = ROLL\_HDG\_HOLD)$

$mode\_Active\_Lateral\_Transition\_Sixteen$

$Transition$

$(\neg\ (term\_SYNC = TRU \land term\_Roll\_LE\_Threshold = TRU)$
$\land\ (term\_SYNC' = TRU \land term\_Roll\_LE\_Threshold' = TRU)$
$\land\ mode\_Active\_Lateral = ROLL\_ROLL\_HOLD$
$\land\ mode\_Active\_Lateral' = ROLL\_HDG\_HOLD)$

$mode\_Active\_Lateral\_Transition\_Seventeen$

$Transition$

$(term\_AP\_Engaged = FALS \land term\_AP\_Engaged' = TRU \land$
$term\_Roll\_LE\_Threshold = TRU \land mode\_Active\_Lateral = ROLL\_ROLL\_HOLD$
$\land mode\_Active\_Lateral' = ROLL\_HDG\_HOLD)$

$mode\_Active\_Lateral\_Transition\_Eighteen$

$Transition$

$(mon\_On\_Ground = FALS \land mon\_On\_Ground' = TRU$
$\land mode\_Active\_Lateral = ROLL\_ROLL\_HOLD$
$\land mode\_Active\_Lateral' = ROLL\_HDG\_HOLD)$

$mode\_Active\_Lateral\_Transition\_Nineteen$

$Transition$

$((mode\_Active\_Lateral \notin ROLL \land mode\_Active\_Lateral' \in ROLL$
$\land term\_Roll\_LE\_Threshold = FALS \land mon\_On\_Ground = FALS)$
$\land mode\_Active\_Lateral = ROLL\_ROLL\_HOLD)$

$mode\_Active\_Lateral\_Transition\_Twenty$

$Transition$

$((mode\_Active\_Lateral = ROLL\_HDG\_HOLD \land$
$\neg\,(term\_SYNC = TRU \land term\_Roll\_LE\_Threshold = FALS \land mon\_On\_Ground = FALS)$
$\land term\_SYNC' = TRU \land term\_Roll\_LE\_Threshold' = FALS \land$
$mon\_On\_Ground' = FALS)$
$\land mode\_Active\_Lateral' = ROLL\_HDG\_HOLD)$

$mode\_Active\_Lateral\_Transition\_TwentyOne$

$Transition$

$((mode\_Active\_Lateral = ROLL\_HDG\_HOLD \land term\_AP\_Engaged = FALS \land$
$term\_AP\_Engaged' = TRU \land term\_Roll\_LE\_Threshold = TRU)$
$\land mode\_Active\_Lateral' = ROLL\_ROLL\_HOLD)$

$mode\_Active\_Lateral\_Transition\_TwentyTwo$

$Transition$

$(mode\_Active\_Lateral = NAV\_ARMED \land$
$\neg\,(Duration\_INMODE\_NAV\_ARMED\_gt\_const\_min\_armed\_period = TRU$
$\land term\_Lateral\_NAV\_Track\_Cond\_Met = TRU) \land$
$Duration\_INMODE\_NAV\_ARMED\_gt\_const\_min\_armed\_period' = TRU \land$
$term\_Lateral\_NAV\_Track\_Cond\_Met' = TRU$
$\land mode\_Active\_Lateral' = NAV\_TRACK)$

$mode\_Active\_Lateral\_Transition\_TwentyThree$

$Transition$

$(((mode\_Active\_Lateral = L\_APPR\_ARMED \wedge$
$(Duration\_INMODE\_APPR\_ARMED\_gt\_const\_min\_armed\_period = FALS \wedge$
$Duration\_INMODE\_APPR\_ARMED\_gt\_const\_min\_armed\_period' = TRU)))$
$\wedge\ mode\_Active\_Lateral' = L\_APPR\_TRACK)$

$\mathit{mode\_Active\_Lateral\_Transition\_TwentyFour}$

$\mathit{Transition}$

$(\neg\,((\mathit{mode\_Flight\_Director} \neq \mathit{FD\_OFF} \land \mathit{mode\_Flight\_Director}' = \mathit{FD\_OFF})$
$\lor\,(\mathit{mode\_Flight\_Director} = \mathit{FD\_OFF} \land \mathit{mode\_Flight\_Director}' \neq \mathit{FD\_OFF} \land$
$\mathit{event} \notin \mathit{Lateral\_Mode\_Requested} \land \mathit{event} \neq \mathit{GA\_Pressed})$
$\lor\,((\mathit{event} = \mathit{HDG\_Switch\_Pressed} \land \mathit{mode\_Active\_Lateral} = \mathit{HDG}))$
$\lor\,((\mathit{event} = \mathit{NAV\_Switch\_Pressed} \land \mathit{mode\_Active\_Lateral} \in \mathit{NAV}))$
$\lor\,((\mathit{event} = \mathit{Nav\_Source\_Changed} \land \mathit{mode\_Active\_Lateral} \in \mathit{NAV}))$
$\lor\,((\mathit{event} = \mathit{APPR\_Switch\_Pressed} \land \mathit{mode\_Active\_Lateral} \in \mathit{L\_APPR}))$
$\lor\,((\mathit{event} = \mathit{Nav\_Source\_Changed} \land \mathit{mode\_Active\_Lateral} \in \mathit{L\_APPR}))$
$\lor\,((\mathit{term\_AP\_Engaged} \neq \mathit{TRU} \land \mathit{term\_AP\_Engaged}' = \mathit{TRU}$
$\land\,\mathit{mode\_Active\_Lateral} = \mathit{L\_GA}))$
$\lor\,((\mathit{term\_SYNC} = \mathit{FALS} \land \mathit{term\_SYNC}' = \mathit{TRU} \land \mathit{mode\_Active\_Lateral} = \mathit{L\_GA}))$
$\lor\,((\mathit{mode\_Active\_Vertical} = \mathit{V\_GA} \land \mathit{mode\_Active\_Vertical}' \neq \mathit{V\_GA} \land$
$\mathit{mode\_Active\_Lateral} = \mathit{L\_GA} \land \mathit{event} \in \mathit{Vertical\_Events} \setminus \mathit{Lateral\_Events}))$
$\lor\,((\mathit{event} = \mathit{HDG\_Switch\_Pressed} \land \mathit{mode\_Active\_Lateral} \neq \mathit{HDG}))$
$\lor\,((\mathit{event} = \mathit{NAV\_Switch\_Pressed} \land \mathit{mode\_Active\_Lateral}' \notin \mathit{NAV}))$
$\lor\,((\mathit{event} = \mathit{APPR\_Switch\_Pressed} \land \mathit{mode\_Active\_Lateral} \notin \mathit{L\_APPR}))$
$\lor\,((\mathit{event} = \mathit{GA\_Pressed} \land \mathit{mode\_Active\_Lateral} \neq \mathit{L\_GA}))$
$\lor\,(\mathit{mode\_Active\_Lateral} \notin \mathit{ROLL} \land \mathit{mode\_Active\_Lateral}' \in \mathit{ROLL}$
$\land\,(\mathit{term\_Roll\_LE\_Threshold} = \mathit{TRU} \lor \mathit{mon\_On\_Ground} = \mathit{TRU}))$
$\lor\,(\neg\,(\mathit{term\_SYNC} = \mathit{TRU} \land \mathit{term\_Roll\_LE\_Threshold} = \mathit{TRU})$
$\land\,(\mathit{term\_SYNC}' = \mathit{TRU} \land \mathit{term\_Roll\_LE\_Threshold}' = \mathit{TRU})$
$\land\,\mathit{mode\_Active\_Lateral} = \mathit{ROLL\_ROLL\_HOLD})$
$\lor\,(\mathit{term\_AP\_Engaged} = \mathit{FALS} \land \mathit{term\_AP\_Engaged}' = \mathit{TRU} \land$
$\mathit{term\_Roll\_LE\_Threshold} = \mathit{TRU} \land \mathit{mode\_Active\_Lateral} = \mathit{ROLL\_ROLL\_HOLD})$
$\lor\,(\mathit{mon\_On\_Ground} = \mathit{FALS} \land \mathit{mon\_On\_Ground}' = \mathit{TRU}$
$\land\,\mathit{mode\_Active\_Lateral} = \mathit{ROLL\_ROLL\_HOLD})$
$\lor\,((\mathit{mode\_Active\_Lateral} \notin \mathit{ROLL} \land \mathit{mode\_Active\_Lateral}' \in \mathit{ROLL}$
$\land\,\mathit{term\_Roll\_LE\_Threshold} = \mathit{FALS} \land \mathit{mon\_On\_Ground} = \mathit{FALS}))$
$\lor\,((\mathit{mode\_Active\_Lateral} = \mathit{ROLL\_HDG\_HOLD} \land$
$\neg\,(\mathit{term\_SYNC} = \mathit{TRU} \land \mathit{term\_Roll\_LE\_Threshold} = \mathit{FALS} \land \mathit{mon\_On\_Ground} = \mathit{FALS})$
$\land\,\mathit{term\_SYNC}' = \mathit{TRU} \land \mathit{term\_Roll\_LE\_Threshold}' = \mathit{FALS} \land$
$\mathit{mon\_On\_Ground}' = \mathit{FALS}))$
$\lor\,((\mathit{mode\_Active\_Lateral} = \mathit{ROLL\_HDG\_HOLD} \land \mathit{term\_AP\_Engaged} = \mathit{FALS} \land$
$\mathit{term\_AP\_Engaged}' = \mathit{TRU} \land \mathit{term\_Roll\_LE\_Threshold} = \mathit{TRU}))$
$\lor\,(\mathit{mode\_Active\_Lateral} = \mathit{NAV\_ARMED} \land$
$\neg\,(\mathit{Duration\_INMODE\_NAV\_ARMED\_gt\_const\_min\_armed\_period} = \mathit{TRU}$
$\land\,\mathit{term\_Lateral\_NAV\_Track\_Cond\_Met} = \mathit{TRU}) \land$
$\mathit{Duration\_INMODE\_NAV\_ARMED\_gt\_const\_min\_armed\_period}' = \mathit{TRU} \land$
$\mathit{term\_Lateral\_NAV\_Track\_Cond\_Met}' = \mathit{TRU})$
$\lor\,(((\mathit{mode\_Active\_Lateral} = \mathit{L\_APPR\_ARMED} \land$
$(\mathit{Duration\_INMODE\_APPR\_ARMED\_gt\_const\_min\_armed\_period} = \mathit{FALS} \land$
$\mathit{Duration\_INMODE\_APPR\_ARMED\_gt\_const\_min\_armed\_period}' = \mathit{TRU})))))$
$\land\,\mathit{mode\_Active\_Lateral}' = \mathit{mode\_Active\_Lateral})$

$$\begin{array}{l}
mode\_Active\_Lateral\_Transition\_Table \mathrel{\widehat=} \\
mode\_Active\_Lateral\_Transition\_One \lor \\
mode\_Active\_Lateral\_Transition\_Two \lor \\
mode\_Active\_Lateral\_Transition\_Three \lor \\
mode\_Active\_Lateral\_Transition\_Four \lor \\
mode\_Active\_Lateral\_Transition\_Five \lor \\
mode\_Active\_Lateral\_Transition\_Six \lor \\
mode\_Active\_Lateral\_Transition\_Seven \lor \\
mode\_Active\_Lateral\_Transition\_Eight \lor \\
mode\_Active\_Lateral\_Transition\_Nine \lor \\
mode\_Active\_Lateral\_Transition\_Ten \lor \\
mode\_Active\_Lateral\_Transition\_Eleven \lor \\
mode\_Active\_Lateral\_Transition\_Twelve \lor \\
mode\_Active\_Lateral\_Transition\_Thirteen \lor \\
mode\_Active\_Lateral\_Transition\_Fourteen \lor \\
mode\_Active\_Lateral\_Transition\_Fifteen \lor \\
mode\_Active\_Lateral\_Transition\_Sixteen \lor \\
mode\_Active\_Lateral\_Transition\_Seventeen \lor \\
mode\_Active\_Lateral\_Transition\_Eighteen \lor \\
mode\_Active\_Lateral\_Transition\_Nineteen \lor \\
mode\_Active\_Lateral\_Transition\_Twenty \lor \\
mode\_Active\_Lateral\_Transition\_TwentyOne \lor \\
mode\_Active\_Lateral\_Transition\_TwentyTwo \lor \\
mode\_Active\_Lateral\_Transition\_TwentyThree \lor \\
mode\_Active\_Lateral\_Transition\_TwentyFour
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{mode\_Active\_Vertical\_Transition\_One} \\
Transition \\
\hline
(mode\_Flight\_Director \neq FD\_OFF \land mode\_Flight\_Director' = FD\_OFF \\
\land\ mode\_Active\_Vertical' = VERTICAL\_NOT\_IN\_MODE) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{mode\_Active\_Vertical\_Transition\_Two} \\
Transition \\
\hline
(mode\_Flight\_Director = FD\_OFF \land mode\_Flight\_Director' \neq FD\_OFF \land \\
event \notin Vertical\_Mode\_Requested \\
\land\ mode\_Active\_Vertical' = PITCH) \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{mode\_Active\_Vertical\_Transition\_Three} \\
Transition \\
\hline
(event = SYNC\_On \land mode\_Active\_Vertical = V\_GA \\
\land\ mode\_Active\_Vertical' = PITCH) \\
\hline
\end{array}$$

$mode\_Active\_Vertical\_Transition\_Four$

$Transition$

$((event = VS\_Pitch\_Wheel\_Changed$
$\land (mode\_Active\_Vertical \in VERTICAL\_MODE \setminus \{VS, V\_APPR, ALTSEL, PITCH\}))$
$\land\ mode\_Active\_Vertical' = PITCH)$

$mode\_Active\_Vertical\_Transition\_Five$

$Transition$

$((\neg (mode\_Altitude\_Select \in ALTSEL\_ACTIVE) \land$
$mode\_Altitude\_Select' \in ALTSEL\_ACTIVE \land mode\_Active\_Vertical \neq ALTSEL)$
$\land\ mode\_Active\_Vertical' = ALTSEL)$

$mode\_Active\_Vertical\_Transition\_Six$

$Transition$

$(term\_Preselected\_Altitude' \neq term\_Preselected\_Altitude \land$
$mode\_Altitude\_Select = ALTSEL\_CAPTURE \land mode\_Active\_Vertical = ALTSEL$
$\land\ mode\_Active\_Vertical' = PITCH)$

$mode\_Active\_Vertical\_Transition\_Seven$

$Transition$

$((term\_Preselected\_Altitude' \neq term\_Preselected\_Altitude \land$
$mode\_Altitude\_Select = ALTSEL\_TRACK \land mode\_Active\_Vertical = ALTSEL)$
$\land\ mode\_Active\_Vertical' = ALTHOLD)$

$mode\_Active\_Vertical\_Transition\_Eight$

$Transition$

$(event = ALT\_Switch\_Pressed \land mode\_Active\_Vertical \notin \{V\_APPR, ALTHOLD\}$
$\land\ mode\_Active\_Vertical' = ALTHOLD)$

$mode\_Active\_Vertical\_Transition\_Nine$

$Transition$

$(event = ALT\_Switch\_Pressed \land mode\_Active\_Vertical = ALTHOLD$
$\land\ mode\_Active\_Vertical' = PITCH)$

$mode\_Active\_Vertical\_Transition\_Ten$

$Transition$

$(event = VS\_Switch\_Pressed \land$
$mode\_Active\_Vertical \notin \{V\_APPR, VS\}$
$\land\ mode\_Active\_Vertical' = VS)$

$mode\_Active\_Vertical\_Transition\_Eleven$

$Transition$

$((event = VS\_Switch\_Pressed \land mode\_Active\_Vertical = VS)$
$\land\ mode\_Active\_Vertical' = PITCH)$

$mode\_Active\_Vertical\_Transition\_Twelve$

$Transition$

$((event = FLC\_Switch\_Pressed \land$
$mode\_Active\_Vertical \notin \{V\_APPR, FLC\_TRACK, FLC\_OVERSPEED\})$
$\land\ mode\_Active\_Vertical' \in FLC)$

$mode\_Active\_Vertical\_Transition\_Thirteen$

$Transition$

$(event = FLC\_Switch\_Pressed \land mode\_Active\_Vertical \in FLC$
$\land\ mode\_Active\_Vertical' = PITCH)$

$mode\_Active\_Vertical\_Transition\_Fourteen$

$Transition$

$(mode\_Active\_Vertical \notin (FLC \cup \{ALTSEL, ALTHOLD, V\_APPR\}) \land$
$term\_Overspeed = FALS \land term\_Overspeed' = TRU$
$\land\ mode\_Active\_Vertical' \in FLC)$

$mode\_Active\_Vertical\_Transition\_Fifteen$

$Transition$

$(mode\_Vertical\_Approach \neq VERT\_APPR\_TRACK$
$\land\ mode\_Vertical\_Approach' = VERT\_APPR\_TRACK$
$\land\ mode\_Active\_Vertical \neq V\_APPR$
$\land\ mode\_Active\_Vertical' = V\_APPR)$

$mode\_Active\_Vertical\_Transition\_Sixteen$

$Transition$

$(\neg\ (event = GA\_Pressed) \land mode\_Vertical\_Approach = VERT\_APPR\_TRACK \land$
$mode\_Vertical\_Approach' \neq VERT\_APPR\_TRACK \land mode\_Active\_Vertical = V\_APPR$
$\land\ mode\_Active\_Vertical' = PITCH)$

$mode\_Active\_Vertical\_Transition\_Seventeen$

$Transition$

$((event = GA\_Pressed \land mode\_Active\_Vertical \neq V\_GA)$
$\land\ mode\_Active\_Vertical' = V\_GA)$

$$\begin{array}{l}
mode\_Active\_Vertical\_Transition\_Eighteen \\
\hline
Transition \\
\hline
((mode\_Active\_Lateral = L\_GA \land mode\_Active\_Lateral' \neq L\_GA \land \\
mode\_Active\_Vertical = V\_GA \land event \in Lateral\_Events \setminus Vertical\_Events) \\
\land mode\_Active\_Vertical' = PITCH)
\end{array}$$

$$\begin{array}{l}
mode\_Active\_Vertical\_Transition\_Nineteen \\
\hline
Transition \\
\hline
((mode\_Active\_Vertical \notin FLC \land \\
mode\_Active\_Vertical' \in FLC \land \neg\ (term\_Overspeed = FALS \land term\_Overspeed' = TRU)) \\
\land mode\_Active\_Vertical' = FLC\_TRACK)
\end{array}$$

$$\begin{array}{l}
mode\_Active\_Vertical\_Transition\_Twenty \\
\hline
Transition \\
\hline
(term\_Overspeed = TRU \land term\_Overspeed' = FALS \land \\
mode\_Active\_Vertical = FLC\_OVERSPEED \\
\land mode\_Active\_Vertical' = FLC\_TRACK)
\end{array}$$

$$\begin{array}{l}
mode\_Active\_Vertical\_Transition\_TwentyOne \\
\hline
Transition \\
\hline
(mode\_Active\_Vertical \notin FLC \land \\
mode\_Active\_Vertical' \in FLC \land term\_Overspeed = FALS \land term\_Overspeed' = TRU \\
\land mode\_Active\_Vertical' = FLC\_OVERSPEED)
\end{array}$$

$$\begin{array}{l}
mode\_Active\_Vertical\_Transition\_TwentyTwo \\
\hline
Transition \\
\hline
(term\_Overspeed = FALS \land term\_Overspeed' = TRU \\
\land mode\_Active\_Vertical = FLC\_TRACK \\
\land mode\_Active\_Vertical' = FLC\_OVERSPEED)
\end{array}$$

$mode\_Active\_Vertical\_Transition\_TwentyThree$

$Transition$

$$
\begin{array}{l}
(\neg\, ((mode\_Flight\_Director \neq FD\_OFF \wedge mode\_Flight\_Director' = FD\_OFF) \\
\vee\, (mode\_Flight\_Director = FD\_OFF \wedge mode\_Flight\_Director' \neq FD\_OFF \wedge \\
event \notin Vertical\_Mode\_Requested) \\
\vee\, (event = SYNC\_On \wedge mode\_Active\_Vertical = V\_GA) \\
\vee\, ((event = VS\_Pitch\_Wheel\_Changed \\
\wedge\, (mode\_Active\_Vertical \in VERTICAL\_MODE \setminus \{VS, V\_APPR, ALTSEL, PITCH\}))) \\
\vee\, ((\neg\, (mode\_Altitude\_Select \in ALTSEL\_ACTIVE) \wedge \\
mode\_Altitude\_Select' \in ALTSEL\_ACTIVE \wedge mode\_Active\_Vertical \neq ALTSEL)) \\
\vee\, (term\_Preselected\_Altitude' \neq term\_Preselected\_Altitude \wedge \\
mode\_Altitude\_Select = ALTSEL\_CAPTURE \wedge mode\_Active\_Vertical = ALTSEL) \\
\vee\, ((term\_Preselected\_Altitude' \neq term\_Preselected\_Altitude \wedge \\
mode\_Altitude\_Select = ALTSEL\_TRACK \wedge mode\_Active\_Vertical = ALTSEL)) \\
\vee\, (event = ALT\_Switch\_Pressed \wedge mode\_Active\_Vertical \notin \{V\_APPR, ALTHOLD\}) \\
\vee\, (event = ALT\_Switch\_Pressed \wedge mode\_Active\_Vertical = ALTHOLD) \\
\vee\, (event = VS\_Switch\_Pressed \wedge \\
mode\_Active\_Vertical \notin \{V\_APPR, VS\}) \\
\vee\, ((event = VS\_Switch\_Pressed \wedge mode\_Active\_Vertical = VS)) \\
\vee\, ((event = FLC\_Switch\_Pressed \wedge \\
mode\_Active\_Vertical \notin \{V\_APPR, FLC\_TRACK, FLC\_OVERSPEED\})) \\
\vee\, (event = FLC\_Switch\_Pressed \wedge mode\_Active\_Vertical \in FLC) \\
\vee\, (mode\_Active\_Vertical \notin (FLC \cup \{ALTSEL, ALTHOLD, V\_APPR\}) \wedge \\
term\_Overspeed = FALS \wedge term\_Overspeed' = TRU) \\
\vee\, (mode\_Vertical\_Approach \neq VERT\_APPR\_TRACK \\
\wedge\, mode\_Vertical\_Approach' = VERT\_APPR\_TRACK \\
\wedge\, mode\_Active\_Vertical \neq V\_APPR) \\
\vee\, (\neg\, (event = GA\_Pressed) \wedge mode\_Vertical\_Approach = VERT\_APPR\_TRACK \wedge \\
mode\_Vertical\_Approach' \neq VERT\_APPR\_TRACK \wedge mode\_Active\_Vertical = V\_APPR) \\
\vee\, ((event = GA\_Pressed \wedge mode\_Active\_Vertical \neq V\_GA)) \\
\vee\, ((mode\_Active\_Lateral = L\_GA \wedge mode\_Active\_Lateral' \neq L\_GA \wedge \\
mode\_Active\_Vertical = V\_GA \wedge event \in Lateral\_Events \setminus Vertical\_Events)) \\
\vee\, ((mode\_Active\_Vertical \notin FLC \wedge \\
mode\_Active\_Vertical' \in FLC \wedge \neg\, (term\_Overspeed = FALS \wedge term\_Overspeed' = TRU))) \\
\vee\, (term\_Overspeed = TRU \wedge term\_Overspeed' = FALS \wedge \\
mode\_Active\_Vertical = FLC\_OVERSPEED) \\
\vee\, (mode\_Active\_Vertical \notin FLC \wedge \\
mode\_Active\_Vertical' \in FLC \wedge term\_Overspeed = FALS \wedge term\_Overspeed' = TRU) \\
\vee\, (term\_Overspeed = FALS \wedge term\_Overspeed' = TRU \\
\wedge\, mode\_Active\_Vertical = FLC\_TRACK)) \\
\wedge\, mode\_Active\_Vertical' = mode\_Active\_Vertical)
\end{array}
$$

$mode\_Active\_Vertical\_Transition\_Table \mathrel{\widehat{=}}$
$mode\_Active\_Vertical\_Transition\_One \lor$
$mode\_Active\_Vertical\_Transition\_Two \lor$
$mode\_Active\_Vertical\_Transition\_Three \lor$
$mode\_Active\_Vertical\_Transition\_Four \lor$
$mode\_Active\_Vertical\_Transition\_Five \lor$
$mode\_Active\_Vertical\_Transition\_Six \lor$
$mode\_Active\_Vertical\_Transition\_Seven \lor$
$mode\_Active\_Vertical\_Transition\_Eight \lor$
$mode\_Active\_Vertical\_Transition\_Nine \lor$
$mode\_Active\_Vertical\_Transition\_Ten \lor$
$mode\_Active\_Vertical\_Transition\_Eleven \lor$
$mode\_Active\_Vertical\_Transition\_Twelve \lor$
$mode\_Active\_Vertical\_Transition\_Thirteen \lor$
$mode\_Active\_Vertical\_Transition\_Fourteen \lor$
$mode\_Active\_Vertical\_Transition\_Fifteen \lor$
$mode\_Active\_Vertical\_Transition\_Sixteen \lor$
$mode\_Active\_Vertical\_Transition\_Seventeen \lor$
$mode\_Active\_Vertical\_Transition\_Eighteen \lor$
$mode\_Active\_Vertical\_Transition\_Nineteen \lor$
$mode\_Active\_Vertical\_Transition\_Twenty \lor$
$mode\_Active\_Vertical\_Transition\_TwentyOne \lor$
$mode\_Active\_Vertical\_Transition\_TwentyTwo \lor$
$mode\_Active\_Vertical\_Transition\_TwentyThree$

$mode\_Altitude\_Select\_Transition\_One$

$Transition$

$(mode\_Flight\_Director \neq FD\_OFF \land mode\_Flight\_Director' = FD\_OFF$
$\land\ mode\_Altitude\_Select' = ALTSEL\_NOT\_IN\_MODE)$

$mode\_Altitude\_Select\_Transition\_Two$

$Transition$

$(mode\_Flight\_Director = FD\_OFF \land mode\_Flight\_Director' \neq FD\_OFF$
$\land\ mode\_Altitude\_Select' = ALTSEL\_ARMED)$

$mode\_Altitude\_Select\_Transition\_Three$

$Transition$

$((mode\_Active\_Vertical \in \{V\_APPR, V\_GA, ALTHOLD\} \land$
$\neg\ (mode\_Active\_Vertical' \in \{V\_APPR, V\_GA, ALTHOLD\})$
$\land\ mode\_Altitude\_Select = ALTSEL\_CLEARED)$
$\land\ mode\_Altitude\_Select' = ALTSEL\_ARMED)$

$mode\_Altitude\_Select\_Transition\_Four$

$Transition$

$((mode\_Active\_Vertical' \in \{V\_APPR, V\_GA, ALTHOLD\} \wedge$
$\neg\, (mode\_Active\_Vertical \in \{V\_APPR, V\_GA, ALTHOLD\}) \wedge$
$mode\_Altitude\_Select \in ALTSEL\_ENABLED)$
$\wedge\; mode\_Altitude\_Select' = ALTSEL\_CLEARED)$

$mode\_Altitude\_Select\_Transition\_Five$

$Transition$

$(mode\_Altitude\_Select = ALTSEL\_ARMED \wedge$
$\neg\, (term\_ALTSEL\_Cond = ALTSEL\_COND\_CAPTURE \wedge$
$Duration\_INMODE\_ALTSEL\_ARMED\_gt\_const\_min\_armed\_period = TRU) \wedge$
$(term\_ALTSEL\_Cond' = ALTSEL\_COND\_CAPTURE \wedge$
$Duration\_INMODE\_ALTSEL\_ARMED\_gt\_const\_min\_armed\_period' = TRU)$
$\wedge\; mode\_Altitude\_Select = ALTSEL\_CAPTURE)$

$mode\_Altitude\_Select\_Transition\_Six$

$Transition$

$((mode\_Active\_Vertical \in \{V\_APPR, V\_GA, ALTHOLD, ALTSEL\} \wedge$
$\neg\, (mode\_Active\_Vertical' \in \{V\_APPR, V\_GA, ALTHOLD, ALTSEL\}) \wedge$
$mode\_Altitude\_Select \in ALTSEL\_ACTIVE)$
$\wedge\; mode\_Altitude\_Select' = ALTSEL\_ARMED)$

$mode\_Altitude\_Select\_Transition\_Seven$

$Transition$

$(mode\_Altitude\_Select = ALTSEL\_CAPTURE \wedge$
$\neg\, (term\_ALTSEL\_Cond = ALTSEL\_COND\_TRACK \wedge$
$Duration\_INMODE\_ALTSEL\_CAPT\_gt\_const\_min\_armed\_period = TRU) \wedge$
$(term\_ALTSEL\_Cond' = ALTSEL\_COND\_TRACK \wedge$
$Duration\_INMODE\_ALTSEL\_CAPT\_gt\_const\_min\_armed\_period' = TRU)$
$\wedge\; mode\_Altitude\_Select' = ALTSEL\_TRACK)$

mode_Altitude_Select_Transition_Eight

Transition

$$
\begin{array}{l}
(\neg\ ((\mathit{mode\_Flight\_Director} \neq \mathit{FD\_OFF} \land \mathit{mode\_Flight\_Director}' = \mathit{FD\_OFF}) \\
\lor (\mathit{mode\_Flight\_Director} = \mathit{FD\_OFF} \land \mathit{mode\_Flight\_Director}' \neq \mathit{FD\_OFF}) \\
\lor ((\mathit{mode\_Active\_Vertical} \in \{\mathit{V\_APPR}, \mathit{V\_GA}, \mathit{ALTHOLD}\} \land \\
\neg\ (\mathit{mode\_Active\_Vertical}' \in \{\mathit{V\_APPR}, \mathit{V\_GA}, \mathit{ALTHOLD}\}) \\
\land \mathit{mode\_Altitude\_Select} = \mathit{ALTSEL\_CLEARED})) \\
\lor ((\mathit{mode\_Active\_Vertical}' \in \{\mathit{V\_APPR}, \mathit{V\_GA}, \mathit{ALTHOLD}\} \land \\
\neg\ (\mathit{mode\_Active\_Vertical} \in \{\mathit{V\_APPR}, \mathit{V\_GA}, \mathit{ALTHOLD}\}) \land \\
\mathit{mode\_Altitude\_Select} \in \mathit{ALTSEL\_ENABLED})) \\
\lor (\mathit{mode\_Altitude\_Select} = \mathit{ALTSEL\_ARMED} \land \\
\neg\ (\mathit{term\_ALTSEL\_Cond} = \mathit{ALTSEL\_COND\_CAPTURE} \land \\
\mathit{Duration\_INMODE\_ALTSEL\_ARMED\_gt\_const\_min\_armed\_period} = \mathit{TRU}) \land \\
(\mathit{term\_ALTSEL\_Cond}' = \mathit{ALTSEL\_COND\_CAPTURE} \land \\
\mathit{Duration\_INMODE\_ALTSEL\_ARMED\_gt\_const\_min\_armed\_period}' = \mathit{TRU})) \\
\lor ((\mathit{mode\_Active\_Vertical} \in \{\mathit{V\_APPR}, \mathit{V\_GA}, \mathit{ALTHOLD}, \mathit{ALTSEL}\} \land \\
\neg\ (\mathit{mode\_Active\_Vertical}' \in \{\mathit{V\_APPR}, \mathit{V\_GA}, \mathit{ALTHOLD}, \mathit{ALTSEL}\}) \land \\
\mathit{mode\_Altitude\_Select} \in \mathit{ALTSEL\_ACTIVE})) \\
\lor (\mathit{mode\_Altitude\_Select} = \mathit{ALTSEL\_CAPTURE} \land \\
\neg\ (\mathit{term\_ALTSEL\_Cond} = \mathit{ALTSEL\_COND\_TRACK} \land \\
\mathit{Duration\_INMODE\_ALTSEL\_CAPT\_gt\_const\_min\_armed\_period} = \mathit{TRU}) \land \\
(\mathit{term\_ALTSEL\_Cond}' = \mathit{ALTSEL\_COND\_TRACK} \land \\
\mathit{Duration\_INMODE\_ALTSEL\_CAPT\_gt\_const\_min\_armed\_period}' = \mathit{TRU}))) \\
\land \mathit{mode\_Altitude\_Select}' = \mathit{mode\_Altitude\_Select})
\end{array}
$$

$$
\begin{array}{l}
\mathit{mode\_Altitude\_Select\_Transition\_Table} \mathrel{\widehat{=}} \\
\mathit{mode\_Altitude\_Select\_Transition\_One} \lor \\
\mathit{mode\_Altitude\_Select\_Transition\_Two} \lor \\
\mathit{mode\_Altitude\_Select\_Transition\_Three} \lor \\
\mathit{mode\_Altitude\_Select\_Transition\_Four} \lor \\
\mathit{mode\_Altitude\_Select\_Transition\_Five} \lor \\
\mathit{mode\_Altitude\_Select\_Transition\_Six} \lor \\
\mathit{mode\_Altitude\_Select\_Transition\_Seven} \lor \\
\mathit{mode\_Altitude\_Select\_Transition\_Eight}
\end{array}
$$

mode_Vertical_Approach_Transition_One

Transition

$$
\begin{array}{l}
(\mathit{mode\_Flight\_Director} \neq \mathit{FD\_OFF} \land \mathit{mode\_Flight\_Director}' = \mathit{FD\_OFF} \\
\land \mathit{mode\_Vertical\_Approach}' = \mathit{VERT\_APPR\_NOT\_IN\_MODE})
\end{array}
$$

mode_Vertical_Approach_Transition_Two

Transition

$$
\begin{array}{l}
(\mathit{mode\_Flight\_Director} = \mathit{FD\_OFF} \land \mathit{mode\_Flight\_Director}' \neq \mathit{FD\_OFF} \\
\land \mathit{mode\_Vertical\_Approach}' = \mathit{VERT\_APPR\_CLEARED})
\end{array}
$$

$mode\_Vertical\_Approach\_Transition\_Three$

$Transition$

$((mode\_Active\_Lateral \neq L\_APPR\_TRACK \land$
$mode\_Active\_Lateral' = L\_APPR\_TRACK \land$
$mode\_Vertical\_Approach = VERT\_APPR\_CLEARED)$
$\land\ mode\_Vertical\_Approach' = VERT\_APPR\_ARMED)$

$mode\_Vertical\_Approach\_Transition\_Four$

$Transition$

$((mode\_Active\_Lateral = L\_APPR\_TRACK \land mode\_Active\_Lateral' \neq L\_APPR\_TRACK$
$\land\ mode\_Vertical\_Approach \in VERT\_APPR\_ENABLED)$
$\land\ mode\_Vertical\_Approach' = VERT\_APPR\_CLEARED)$

$mode\_Vertical\_Approach\_Transition\_Five$

$Transition$

$(mode\_Vertical\_Approach = VERT\_APPR\_ARMED \land$
$\neg\ (term\_Vertical\_APPR\_Track\_Cond\_Met = TRU \land$
$Duration\_INMODE\_Vert\_Appr\_Track\_gt\_const\_min\_armed\_period = TRU) \land$
$(term\_Vertical\_APPR\_Track\_Cond\_Met' = TRU \land$
$Duration\_INMODE\_Vert\_Appr\_Track\_gt\_const\_min\_armed\_period' = TRU)$
$\land\ mode\_Vertical\_Approach' = VERT\_APPR\_TRACK)$

$mode\_Vertical\_Approach\_Transition\_Six$

$Transition$

$(\neg\ ((mode\_Flight\_Director \neq FD\_OFF \land mode\_Flight\_Director' = FD\_OFF)$
$\lor\ (mode\_Flight\_Director = FD\_OFF \land mode\_Flight\_Director' \neq FD\_OFF)$
$\lor\ ((mode\_Active\_Lateral \neq L\_APPR\_TRACK \land$
$mode\_Active\_Lateral' = L\_APPR\_TRACK \land$
$mode\_Vertical\_Approach = VERT\_APPR\_CLEARED))$
$\lor\ ((mode\_Active\_Lateral = L\_APPR\_TRACK \land mode\_Active\_Lateral' \neq L\_APPR\_TRACK$
$\land\ mode\_Vertical\_Approach \in VERT\_APPR\_ENABLED))$
$\lor\ (mode\_Vertical\_Approach = VERT\_APPR\_ARMED \land$
$\neg\ (term\_Vertical\_APPR\_Track\_Cond\_Met = TRU \land$
$Duration\_INMODE\_Vert\_Appr\_Track\_gt\_const\_min\_armed\_period = TRU) \land$
$(term\_Vertical\_APPR\_Track\_Cond\_Met' = TRU \land$
$Duration\_INMODE\_Vert\_Appr\_Track\_gt\_const\_min\_armed\_period' = TRU)))$
$\land\ mode\_Vertical\_Approach' = mode\_Vertical\_Approach)$

$mode\_Vertical\_Approach\_Transition\_Table \mathrel{\widehat=}$
$mode\_Vertical\_Approach\_Transition\_One \lor$
$mode\_Vertical\_Approach\_Transition\_Two \lor$
$mode\_Vertical\_Approach\_Transition\_Three \lor$
$mode\_Vertical\_Approach\_Transition\_Four \lor$
$mode\_Vertical\_Approach\_Transition\_Five \lor$
$mode\_Vertical\_Approach\_Transition\_Six$

```
┌─ Transition_Tables ─────────────────────────────
│ mode_Overspeed_Transition_Table
│ mode_Autopilot_Transition_Table
│ mode_Flight_Director_Transition_Table
│ mode_Active_Lateral_Transition_Table
│ mode_Active_Vertical_Transition_Table
│ mode_Altitude_Select_Transition_Table
│ mode_Vertical_Approach_Transition_Table
└─────────────────────────────────────────────────
```

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** January 1998	**3. REPORT TYPE AND DATES COVERED** Contractor Report

4. TITLE AND SUBTITLE Formal Specification of a Flight Guidance System	**5. FUNDING NUMBERS** C NAS1-20335 WU 519-30-31-01
6. AUTHOR(S) Francis Fung and Damir Jamsek	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Odyssey Research Associates, Cornell Business & Technology Park, 33 Thornwood Drive, Suite 500, Ithaca, NY 14850-1250	**8. PERFORMING ORGANIZATION REPORT NUMBER** TR 97-0042
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration, Langley Research Center, Hampton, VA 23681-2199	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** NASA/CR-1998-206915

11. SUPPLEMENTARY NOTES
NAS1-20335 Task 8 Final Report
Langley Technical Monitor: Ricky W. Butler

12a. DISTRIBUTION / AVAILABILITY STATEMENT Unclassified-Unlimited; Subject Categories 59, 61; Distribution: Nonstandard; Availability: NASA CASI (301) 621-0390	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 words)***

In this report, we investigate the use of formal methods on the semi-formal specification of a Flight Guidance System described in Steven P. Miller and Karl F. Hoech's "Specifying the Mode Logic of a Flight Guidance System in CoRE." The CoRE method can be used with the formal semantics of the SCR discrete-time formal model. However, Miller and Hoech's specification does not satisfy the restrictions of this formal model; for instance, they use concurrent mode machines that drive each other. Furthermore, Miller and Hoech add several notions to CoRE in their specification, without formal definitions.

We use the Z notation to give a fornal semantics for the Flight Guidance System, by adapting the SCR formal model's definitions of event and transitions, which mesh well with Z conventions. In particular, we do not define any micro-time semantics. We give formal definitions for Miller and Hoech's extensions to CoRE, expect for the "continuous transition to FLC," which seems to be best expressed using micro-time semantics. We perform experiments of formal verification on the specification using Z/EVES. In restricted versions of the specification, we are able to do table consistency checking and to verify properties such as determinism and system invariants.

14. SUBJECT TERMS formal methods, Z notation, CoRE methods, SCR formal model, flight guidance system	**15. NUMBER OF PAGES** 101
	16. PRICE CODE A06

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT**